Anderson dos Santos Galeote

Uma aplicação do framework Apache Isis

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para conclusão do curso de MBA em Tecnologia de Software.

São Paulo
2014

Anderson dos Santos Galeote

Uma aplicação do framework Apache Isis

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a conclusão do curso de MBA em Tecnologia de Software.

Área de Concentração: Tecnologia de Software

Orientadora: Prof. Dra. Jussara Pimenta Matos

São Paulo
2014

DEDICATÓRIA

Dedico este trabalho à minha esposa, que pacientemente me apoiou durante todo este curso, e que sempre me deu forças para continuar crescendo profissionalmente.

AGRADECIMENTOS

Ao PECE – Programa de Educação Continuada em Engenharia que forneceu neste período de aulas do curso de especialização em Tecnologia de Software, grandes conhecimentos, qualidade e experiência de seu quadro de professores.

À professor Jussara, que pacientemente revisou meu trabalho e soube lidar com as condições adversas do trabalho, dando conselhos e corrigindo de maneira direta e assertiva.

Aos meus pais e irmãos que sempre me apoiaram para ingressar na pós graduação para complementar meus conhecimentos e minha formação.

E principalmente a minha esposa, por além de me incentivar, apoiar e servir de conselheira, é a responsável por me dar forças para lutar e continuar indo atrás dos meus sonhos.

RESUMO

Este trabalho apresenta uma aplicação prática da ferramenta Apache Isis, disponível como projeto open source, e utilizada para geração rápida de aplicações am Java baseadas em modelo de negócio. Este modelo utiliza o DDD (Domain Driven Design) e o paradigma Naked Objects. Para o desenvolvimento deste tema, foi estudada uma reconstrução inicial de um sistema real de uma pequena empresa do ramo logístico, utilizando a ferramenta Apache Isis na fase de prototipação e o DDD na especificação técnica.

Palavras-chave: Domain Driven Design, Naked Objects, Apache Isis, Java.

ABSTRACT

This work presents a practical application of Apache Isis, available as a open source project, and used for rapid development of applications in java, through business model using DDD (Domain Driven Design) and the paradigm Naked Objects. To do so, it will be evaluated a initial rebuild of a real application owned by a small logistics company, usind Apache Isis tool for prototyping and DDD in the technical specification.

Keywords: Domain Driven Design, Naked Objects, Apache Isis, Java.

LISTA DE ILUSTRAÇÕES

Pág.

Figura 1 – Tela do Framework Naked Objects aplicado ao Carserv em interface Client Server 23
Figura 2 – Tela do Framework Naked Objects aplicado ao Carserv em interface HTML 23
Figura 3 – Exemplo significado da arquitetura hexagonal com algumas portas "ativas" 25
Figura 4 – Tela de listagem mostrando a lista de To Dos (Tarefas) de forma direta 26
Figura 5 – ToDoItems herda da classe AbstractFactoryAndRepository 27
Figura 6 – Menu do Isis trazendo as ações possíveis, que estão implementadas na classe ToDoItems do tipo AbstractFactoryAndRepository 28
Figura 7 – Tela de cadastro de uma nova tarefa ("ToDo") 28
Figura 8 – Tela de detalhe de uma nova tarefa ao se clicar em algum item da listagem da primeira imagem 29
Figura 9 – Tela de relatório de viagens 40
Figura 10 – Tela de listagem de veículos 40
Figura 11 – Cadastro de veículos 41
Figura 12 – Diagrama de classes com aplicação de modelo de domínio 42
Figura 13 – Tela da listagem de viagens prototipada via Apache Isis 44
Figura 14 – Tela de edição de uma viagem 45
Figura 15 – Tela de listagem de veículos 45

LISTA DE TABELAS

Pág.

Tabela 1 – Tabela retirada da sessão 7.4 da tese de Pawson (2004)..... 36
Tabela 2 – Comparação dos índices acerca de classes externas para cada estudo de caso..... 37

LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous Javascript and XML
API	Application Programming Interface
ASP	Active Server Pages
DDD	Domain Driven Design
DSFA	Department of Social and Family Affairs
DSL	Domain Specific Language
HTML	Hyper-Text Markup Language
IDE	Integrated Development Environment
JDO	Java Data Objects
JPA	Java Persistence API
MDD	Model Driven Design
MVC	Model View Controller
POJO	Plain Old Java Objects
XML	Extensible Markup Language

SUMÁRIO

Pág.

1. INTRODUÇÃO 11
1.1 Motivações 11
1.2 Objetivo 12
1.3 Justificativas 12
1.4 Estrutura do Trabalho 13
2. REVISÃO BIBLIOGRÁFICA 15
2.1 Domain Driven Design 15
2.2 Naked Objects – Paradigma e framework 19
2.3 Framework Apache Isis 22
2.4 Especificaçao do Apache Isis 25
3. Proposta de Evolução de Software 30
3.1 Apresentação das deficiências do software atual da transportadora 30
3.2 Domain Driven Design e Isis 31
3.3 Enriquecimento do modelo de negócio 32
3.4 Framework orientado a objetos compatível com o DDD 32
4. Avaliação dos resultados 35
4.1 Avaliação do Naked Objects 35
4.2 Aplicação da proposta para o software de controle logístico atual não Orientado a Objetos – Apache Isis em ação 38
4.3 Domain Driven Design aplicado ao problema 41
4.4 Naked Objects/Apache Ísis aplicado ao problema 43
4.5 Avaliação dos Resultados 47
4.5.1 Evolução da especificação 47
4.5.2 Prototipação e Fase Inicial 47
4.5.3 Comparação da implementação 48
4.5.4 Considerações do Capítulo 49
5. CONSIDERAÇÕES FINAIS 51
5.1 Contribuições do Trabalho 51
5.2 Trabalhos Futuros 52
REFERÊNCIAS 53

1. INTRODUÇÃO

1.1 Motivações

O mercado de tecnologia da informação evolui de forma incessante, desde o surgimento da computação e da sua adoção em massa. Esta evolução colabora para que cada vez surjam novas ideias, novas formas de atuação e novos conceitos.

Numa área como a da tecnologia da informação, é crítico o fato de que os profissionais que se dedicam a ela, estudem não apenas a esfera de conhecimento necessária para a execução de seu trabalho comercial ou evolua em sua pesquisa científica, mas também que transcenda esta esfera de conhecimento, buscando sempre aprender e melhorar a qualidade do trabalho executado.

Diante desta evolução de conceitos e de formas de se pensar, é interessante para todos os envolvidos, tais como, instituições de ensino, laboratórios de inovação, empresas privadas investidoras em tecnologia, empresas consumidoras de tecnologia, profissionais do mercado de trabalho, que haja novos estudos e novas descobertas nesta área, capazes de impactar positivamente a forma como é aplicada a tecnologia no dia a dia.

O desenvolvimento de software é um tema que normalmente é estudado desde a sua concepção até a manutenção, em todo o ciclo de vida. Melhorias nesta área trazem um avanço significativo na maneira como se aplica software e como se resolvem problemas diante de um mundo cada vez mais tecnológico. O tema abordado referente ao desenvolvimento de aplicações oferece um aprofundamento que o autor gostaria de desenvolver, principalmente utilizando técnicas provenientes de pesquisa que mostram ser influenciadoras de tecnologias de mercado largamente usadas.

Ao se estudar uma ferramenta cujo uso está em crescimento por alguns anos, aliada ao conhecimento adquirido em desenvolvimento de aplicações que utilizam código aberto, é possível contribuir com a experiência. Isto pode ser alcançado, seja compreendendo melhor a ferramenta proposta através do estudo da parte teórica, ou participando ativamente da comunidade que a mantém e contribuindo com algum tipo ação.

1.2 Objetivo

O objetivo principal deste estudo é aplicar ferramentas para atualizar um software existente, estudando possíveis melhorias que podem ser adquiridas nas fases iniciais de desenvolvimento, com o foco no enriquecimento de detalhes na especificação do modelo de negócios, utilizando a notação UML, e também no processo de prototipação da aplicação.

Para um estudo envolvendo a atividade de desenvolvimento, é necessário que sejam explorados conceitos técnicos que possam ilustrar o contexto no qual ele está inserido, e encaixá-lo de forma a ser localizado entre o que se tem hoje disponível no mercado de TI, como uma técnica de desenvolvimento, ou uma aplicação de uma ferramenta.

Nesta monografia, a aplicação de uma ferramenta de código aberto é proposta e executada, unindo conceitos já conhecidos e consolidados, com novas ideias que tem como objetivo aprimorar o processo de desenvolvimento de sistemas. Este é o caso dos conceitos do Domain Driven Design (EVANS, 2004) e do Naked Objects (PAWSON, 2002), além da ferramenta Apache Isis (HAYWOOD,2013).

1.3 Justificativas

Atualmente a orientação a objetos é um dos paradigmas com extensa adoção do mercado de TI. Existem diversas outras ferramentas, linguagens e tecnologias que estão relacionadas entre si, e movimentam a atividade da construção de software. Em particular, as tecnologias de código fonte aberto e livres, atraem pessoas de diversas parte do mundo, interessadas em solidificar esses projetos e ainda mais contribuir intelectualmente com a proposta.

O Apache Isis (HAYWOOD,2013) é uma dessas ferramentas, que está em um processo tímido de avanço, porém tem uma comunidade ativa que apoia a evolução do produto e que busca difundir a utilização de seus conceitos. Sua popularidade se deve aos anos de pesquisa em que o Naked Objects (PAWSON, 2002) está presente desde as primeiras obras de que o apresentaram.

A ferramenta traz consigo a utilização da tecnologia Java, atualmente uma das linguagens de programação mais utilizadas, aplica a orientação a objetos, e está

em um âmbito de projeto de código fonte aberto, fazendo parte da organização Apache, que é respeitada pela força que tem neste nicho de mercado colaborativo.

Além disso, propõe uma quebra de paradigmas ao se trazer inovações na maneira como se projeta e se constrói software, com a premissa de beneficiar todos envolvidos em seu contexto de aplicação, além de trazer avanços no conhecimento dos desenvolvedores e especialistas do projeto para uma maior utilização de conceitos de modelagem e a utilização de uma linguagem poderosa como Java para a criação de aplicações de qualidade em relação à expectativa dos usuários.

A utilização do Domain Driven Design é um ponto crucial para o estudo e a aplicação de um estudo do Apache Isis. Portanto, o trabalho utiliza destes conceitos para fornecer subsídios para o Apache Isis, e desta forma proporcionar um contexto ideal de utilização.

Estudar conceitos e ferramentas com estas características, que abrange fases cruciais no projeto de desenvolvimento de software, é uma valiosa oportunidade para um profissional que estuda e atua com a Engenharia de Software.

1.4 Estrutura do Trabalho

O Capítulo 1 apresenta as motivações, o objetivo, as justificativas e a condução do trabalho quanto às ferramentas e conceitos que são estudados, e principalmente o que levou o autor a pesquisar sobre os temas.

O Capítulo 2 apresenta os conceitos que foram necessários para a aplicação do framework Apache Isis, que incluem o paradigma Naked Objects, e o Domain Driven Design.

O Capítulo 3 apresenta a proposta da adoção das técnicas visando promover melhorias no desenvolvimento e atualização de um sistema legado, através da evolução de código procedural para uma modelagem Orientada a Objetos, enriquecendo o modelo de negócios e o protótipo do produto novo.

O Capítulo 4 apresenta dois estudos de caso, um deles sendo a apresentação de um estudo presente nas referências técnicas. O outro é o ponto chave da aplicação

do trabalho, que coloca em prática a modelagem Orientada a Objetos utilizando o Domain Driven Design e aplica a prototipação com o Apache Isis. Ao final deste, demonstra-se a avaliação dos resultados qualitativos perante a utilização das técnicas e ferramentas deste trabalho na atualização do software.

O Capítulo 5 descreve as conclusões referente ao estudo proposto e aplicado, e apresenta um resumo sobre o processo de conhecimento, estudo e aplicação principalmente do Apache Isis. Apresenta-se neste capítulo os próximos passos que poderiam ser dados para se continuar o estudo da ferramenta.

REFERÊNCIAS relacionam os principais autores explorados no trabalho, com maior destaque para as obras que serviram de base para a aplicação do *framework*, como a obra que apresenta o DDD (EVANS, 2004), as obras que apresentam o Naked Objects (PAWSON, 2004) (HAYWOOD, 2009) e a apresentação do Apache Isis (HAYWOOD, 2013).

2. REVISÃO BIBLIOGRÁFICA

Este trabalho aborda o paradigma DDD (Domain Driven Design) (EVANS, 2004), que inclui padrões de projeto para a composição de seus conceitos, o paradigma Naked Objects (PAWSON, 2002) e o framework Apache Isis (HAYWOOD, 2013), ambos baseados na geração de aplicações orientadas a objetos.

2.1 Domain Driven Design

Na área de Desenvolvimento de Software, o paradigma de Orientação a Objetos pode se beneficiar com a adoção de Padrões de Projetos ("Design Patterns") (FOWLER, 2002). Estes padrões são utilizados e desenvolvidos para facilitar e organizar o desenvolvimento, padronizando aplicação de conceitos que acabam se repetindo em diferentes projetos.

Autores e pesquisadores (CARMICHAEL, 2002; FOWLER,1999) têm se dedicado a encontrar formas de desenvolver software de maneira facilitada, principalmente nas fases de implementação, procurando otimizar o tempo e por consequência diminuir o custo. Além disso, facilitar a compreensão do software tanto em tempo de projeto quanto em tempo de manutenção.

O DDD (EVANS, 2004) é um conjunto de práticas, processos e técnicas de modelagem do negócio, que propõe principalmente aproximar o modelo de négocio da implementação do software. O objetivo é prover uma forma de abordar o modelo explicitando de forma direta as regras do processo a ser implementado, utilizando-se dos Padrões de Projeto, iterações rápidas e assertivas em relação às regras de negócio e desta forma, gerar um software não apenas que atenda as características inicialmente requisitadas, mas intrisecamente representar o processo de forma fiel. Iterações ágeis são ciclos encurtados de desenvolvimento e entrega de software de forma incremental, norteados pelo envolvimento e aprovação do cliente a cada uma destas entregas.

Um item importante apresentado no Domain Driven Design, é a caracterização de uma linguagem ubíqua ("Ubiquitous Language") (EVANS,2004). Essa linguagem única pode ser definida como o dialeto a ser especificado e adotado

por todos envolvidos no processo de especificação do negócio, sejam eles analistas de negócio, desenvolvedores, gerentes e usuários.

Evans (2004) propõe que para se iniciar um processo de evolução na modelagem, deve ocorrer um *brainstorm* com os interessados no projeto em suas fases iniciais, bem como uma padronização na linguagem do negócio criando um vocabulário de projeto. Desta forma, unirá o que o usuário tem de conhecimento, traduzido para uma linguagem única que poderá representar diretamente as regras em um diagrama de classes, por exemplo, ou em qualquer modelo capaz de representar os processos e particularidades do negócio. Isto tem como resultado a composição da Linguagem Ubíqua, que é uma linguagem única e comum entre todos envolvidos no projeto, estabelecida para facilitar o entendimento do modelo de negócio. Em Evans (2004) é apresentada uma definição direta e simples: *"Com um esforço consciente entre a equipe, o modelo de domínio pode prover a espinha dorsal para essa linguagem comum".*

O Domain Driven Design propõe técnicas e práticas que permitem uma melhor tradução do cenário real. Para os primeiros passos da exploração do conceito, são apresentadas algumas dessas técnicas chave para o início do processo de levantamento de requisitos, sendo elas: Entidade, Objeto de Valor, Serviço, Agregação, Fábrica e Repositório.

Para iniciar, é aconselhável que se elabore um trabalho conjunto entre detentores do conhecimento e especialistas técnicos responsáveis pela implementação para que o conhecimento se unifique e se firme. Não existe também obrigatoriedade no conceito de modelo a ser usado para representação, porém para o DDD, assume-se o diagrama de classes proposto na especificação da UML (BOOCH, 2005), desenvolvida como uma das formas para se representar modelos orientados a objeto e sistêmicos.

As principais técnicas e padrões que são explorados no Domain Driven Design são:

A) Entidade: o principal padrão de projeto utilizado para definir objetos no DDD. A entidade, também conhecida como objeto de referência, pode ser determinada quando um objeto é idealizado e modelado ao se levar em consideração que o mesmo tem uma identidade única. Para o DDD, as entidades são os principais objetos. De preferência, devem ser os primeiros a serem identificados junto aos usuários que conhecem o modelo.

B) Objetos de Valor: diferente da Entidade, é possível utilizar de diversos outros objetos na modelagem da negócio, que não possuem uma identidade conceitual, chamados de Objetos de Valor. Esses objetos por vezes podem ser confundidos com características de Entidades, sendo campos primitivos de uma dada tecnologia, como um inteiro, ou uma estrutura de texto. Porém, ao levar em consideração o modelo de negócio que é alvo de análise, por vezes é interessante que seja utilizada uma estrutura de objeto que irá ter suas características.

C) Serviço: quando o modelo possui alguma necessidade de negócio, que não necessariamente pode ser satisfeita pelas Entidades ou pelos Objetos de Valor, e uma vez que essa necessidade seja para todo o modelo, pode ser considerada a criação de Serviços. O autor do DDD (EVANS, 2004) sugere os seguintes itens relevantes a serem levados em conta para uma criação de um Serviço adequado:

- A operação identificada no modelo de negócio se relaciona com um conceito que não é uma parte natural nem de uma Entidade ou de um Objeto de Valor;
- A interface é definida em termos de outros elementos do modelo de domínio;
- A operação é independente de estado do objeto.

D) Agregação: com base nos conceitos do DDD, existem situações aonde é recomendada a utilização de um processo de organização de entidades e objetos de valor do modelo, no qual se agregam os objetos dentro de um conceito. Este conceito visa manter a consistência das associações entre as classes, de forma que a alteração em uma agregação aconteça de forma controlada e coordenada pelo objeto principal. Neste caso, o objeto principal é chamado de raiz da agregação.

E) Fábrica: para a construção de um objeto com maior complexidade, o DDD incorpora o padrão de Fábrica. A Fábrica deve ser utilizada para criação de objetos Agregados, cuja constituição pode requerer operações coordenadas, e que sejam realizadas de acordo com o conjunto de objetos que o compõe, levando em consideração como cada um destes componentes deve ser encaixado ao objeto

Raíz. A Fábrica possui a receita para a construção de um produto a ser solicitado pelo código.

F) Repositório: outro conceito apresentado pelo autor do DDD (EVANS, 2004) é o de Repositório. Este conceito visa fornecer uma interface com o mecanismo de provisionamento de dados. Este mecanismo pode ser um banco de dados relacional, arquivos físicos, arquivos xml ou texto, e até dados na memória. O principal objetivo, é criar um Repositório que forneça métodos de adição, alteração, e remoção de itens, bem como métodos de acesso e composição de critério para que o mesmo devolva um objeto ou uma coleção de objetos. Em caso de agregação, deve ser considerado o objeto raíz que seja de relevância, e a consistência entre seus agregados. As classes repositório têm uma ligação estreita com as classes do tipo Fábrica, uma vez que a última precisa de dados que normalmente estão em um mecanismo de armazenamento específico, funcionalidade aplicada pelo Repositório.

Estas são algumas das técnicas apresentadas na obra original e que são adotadas no escopo deste trabalho. Elas podem ser combinadas de forma a melhorar o modelo na tentativa de deixá-lo da forma mais coerente com a regra de negócio. Inclusive, o autor exercita e aplica técnicas de refatoração de um modelo de exemplo em sua obra (EVANS, 2004), demonstrando a importância no uso das mesmas para esta fase de especificação. Ou seja, um dos processos mais importantes no qual o DDD se apoia e incentiva, é a constante exploração do modelo com os detentores de conhecimento, sempre procurando prover maior clareza, consistência e maturidade.

Além da utilização destes padrões, a formação de uma DSL (Domain Specific Language) também é proposta por Evans (2004). Esta proposta seria, escrever e especificar de forma clara e informativa já em linhas de código, o que cada trecho do software está se propondo a executar. Isso pode ser alcançado ao se dar nomes objetivos para variáveis e usar uma sequência descritiva como nome de funções.

Desta forma, o próprio código passa a comunicar as regras do negócio, e desta forma facilitar o entendimento deste código para desenvolvedores ou analistas novos, e até mesmo para um pessoas sem conhecimento técnico na linguagem de

programação. A DSL deve concretizar os detalhamentos da linguagem única na escrita do código fonte.

2.2 Naked Objects – Paradigma e framework

O Naked Objects foi proposto por Richard Pawson e Robert Mathews (2002) oficialmente na obra de mesmo nome. Foi criado como uma proposta nova referente ao paradigma orientado a objetos, e que vem para mudar a forma como hoje são empregadas as técnicas de levantamento de requisitos e a especificação do software desde a fase do protótipo, até as iterações de construção do mesmo.

Os autores em sua obra buscam mostrar que atualmente existe um conceito mal utilizado da orientação a objetos e propõem uma nova abordagem para tal. O principal quesito citado frequentemente pelo autor é a contínua separação entre comportamento e dados, indo contra os princípios da própria orientação a objetos. Em sua obra cita uma colocação sobre esta separação, definindo como "(...) acoplamento excessivo, e uma distribuição inadequada da inteligência da aplicação entre as classes(...)" (FIRESMITH, 1996).

Pawson (2004) publicou posteriormente em sua tese de doutorado, o paradigma e o framework orientado a objetos, ambos chamados de Naked Objects, além de estudos que comprovassem seu ponto de vista sobre a orientação a objetos. Este paradigma é definido como uma representação direta de objetos, modelados de forma a possuírem uma verdadeira "completeza de comportamento" e responsabilidade. Esta representação é feita através de uma interface gráfica orientada a objetos (OOUI – Object Oriented User Interface) (COLLINS, 1995), trazendo para o usuário os objetos como unidades disponíveis para manipulação e ativação de ações.

Diferente de interfaces hoje conhecidas, o usuário deixa de operar sobre tarefas sequenciais para obter um cenário de execução, mas sim uma interação entre os objetos que fazem sentido para o processo de negócio, corretamente relacionados a um alto nível de coesão entre classes e também com o processo mental para a resolução de problemas.

Para comprovar e dar fundamento a suas idéias, os criadores Pawson e Matthews (2002), exploram conceitos propostos desde a década de 60, período considerado marco da criação das primeiras tentativas de representação de software

através de objetos. Procuram mostrar que através do tempo, houve um desvio do que se entende por software orientado a objeto, e da forma como o paradigma é aplicado no design e na implementação.

Por vezes, o modelo de negócio é empobrecido ao ponto de ser identificável apenas pelos nomes em comum de entidades, e com responsabilidades disseminadas por N camadas, criando muitas instâncias da mesma entidade em cada camada, com responsabilidades diferentes. Um exemplo disso seria nas aplicações modernas do conceito MVC (Model-View-Controller) (REENSKAUG, 1979).

Para o modelo MVC, existem objetos que possuem responsabilidades na camada de Modelo (comumente utilizada para toda a lógica do negócio e interface com a persistência de dados), na camada Controller (normalmente, objetos que saberão como orquestrar o fluxo entre o que o usuário faz na interface com o computador e o que será executado dentro do sistema) e até uma tela da camada de View (exemplo de uma página em um browser ou uma tela, que é responsável pela interface com o usuário).

Qualquer alteração no contexto de um objeto como um novo atributo e regra de manipulação do mesmo, poderia levar a uma manutenção disseminada em vários artefatos. O contexto deste objeto estaria fragmentado, quebrando um entendimento fluente de quais são os limites de sua utilização e responsabilidades dentro do código fonte. Um desenvolvedor novo na equipe acabaria entendendo primariamente como um determinado conceito funciona para a implementação do software, antes mesmo de entender as regras do negócio corretamente.

Em sua tese, Pawson (2004) teve o privilégio de ter seu trabalho revisado pelo professor Trygve Reenskaug, criador do modelo Model View Controller (MVC). Conhecido como um dos pioneiros da programação orientada a objetos, o próprio publica na revisão do trabalho de Pawson (2004), que seu objetivo ao criar o modelo MVC era prover um maior poder ao usuário na resolução de problemas ao usar o software, tendo este uma tradução fiel de como a estrutura está na cabeça do próprio utilizador. A camada de Modelo foi visualizada como a camada mais importante, que traria a representação deste modelo de forma contundente para dentro do software. Porém, posteriormente houve uma alteração do conceito, como descrito pelo professor Reenskaug:

> [...] O MVC original foi alterado posteriormente no Smalltalk-80 para se tornar uma solução que separou entrada, saída e informação. O participante mais importante na arquitetura MVC original, a mente do usuário, foi de alguma forma esquecida. (...) Existem duas tradições em aplicações de computadores; uma é empregar o computador para potencializar seus usuários, e outra é aplicar o computador para controlar os usuários. Eu lamento dizer que a última parece ser prevalecedora na área dominante da computação de hoje. [...] (REENSKAUG, 2004, p. 3)

No Naked Objects, a utilização do conceito de MVC não é abolida, mas sim sugere-se que as camadas de View e Controller devem se tornar um poderoso conjunto de transformação de objetos da camada Model, em uma interface OOUI, colocando em prática o conceito de objetos expostos ao usuário, e de potencialização de suas ações através do software.

Outro conceito discutido e criticado na tese, é a utilização da técnica de levantamento de requisitos utilizando-se casos de uso. Desta vez, são citados autores como Firesmith (1996), que defende que o caso de uso quando usado para a identificação de objetos na fase de exploração do software, pode causar um empobrecimento na criação do modelo levando a um alto índice de acoplamento entre objetos, e ainda mais podendo disseminar as características e responsabilidades dos objetos entre as diversas camadas a serem criadas. Isto, devido a importância dada a sequência de ação do usuário já se pensando na interação entre homem e computador, e esquecendo as importantes interações e relacionamentos entre as entidades identificadas.

Uma evidência disso, é o mapeamento dos passos descritos em um caso de uso, para um objeto controlador ou orquestrador, que deve saber como a aplicação fará todo o fluxo envolvendo os objetos, o que atrapalharia a completeza dos objetos do domínio defendida por Pawson (2004). Entretanto, em sua obra, existe o reconhecimento que casos de uso são interessantes ao serem usados para validação de protótipo ou até de resultado de uma ou mais iterações de desenvolvimento, ao se ter um dos cenários completamente construído no software. O caso de uso pode servir como um poderoso guia de testes.

Um dos principais pontos citados por Richard Pawson (2004), seria a ressalva ao indicar o tipo de projeto que se encaixa melhor com o paradigma Naked Objects.

Para uma avaliação desta questão, deve-se verificar se qualquer uma das sentenças abaixo é verdadeira:

- Haverá benefício em caracterizar o papel do usuário como um resolvedor de problemas, ao invés de um seguidor de processo;
- Futura agilidade do negócio é uma preocupação primária;
- Os requisitos são incertos.

E se todas as sentenças abaixo são verdadeiras:

- Não há nenhuma razão clara para se ter uma interface personalizada feita "a mão";
- Os usuários serão usuários frequentes;
- Todo processamento batch é relativamente simples, ou pode ser tratado como um sistema separado.

Além da apresentação da teoria do que seria o Naked Objects, Pawson (2004) demonstra estudos que indicam a viabilidade na adoção do conceito em projetos reais. A criação do framework que implementa o conceito evoluiu nos projetos que o autor executou em projetos que conduziu.

2.3 Framework Apache Isis

O framework Apache Isis tem como seu embrião o projeto ainda conduzido como Naked Objects, iniciado por Pawson e Matthews (2002). Apresentado também por Haywood (2009) que possuía inicialmente duas interfaces disponíveis para estudo: uma principal com o foco em aplicações client-server, e outra para aplicações Web.

Em sua obra, como parte da condução do estudo da aplicação, Haywood propõe um projeto de uma aplicação fictícia de uma oficina mecânica de carros chamada CarServ (este modelo de projeto já era um exemplo utilizado em um livro anterior de Dan Haywood e Andy Carmichael (2002) e também aplicado na tese de Pawson (2002)). Seguem duas imagens, figuras 1 e 2, que mostram como seriam as telas do CarServ para cada objetivo:

Figura 1 - Tela do framework Naked Objects aplicado ao CarServ em interface Client Server

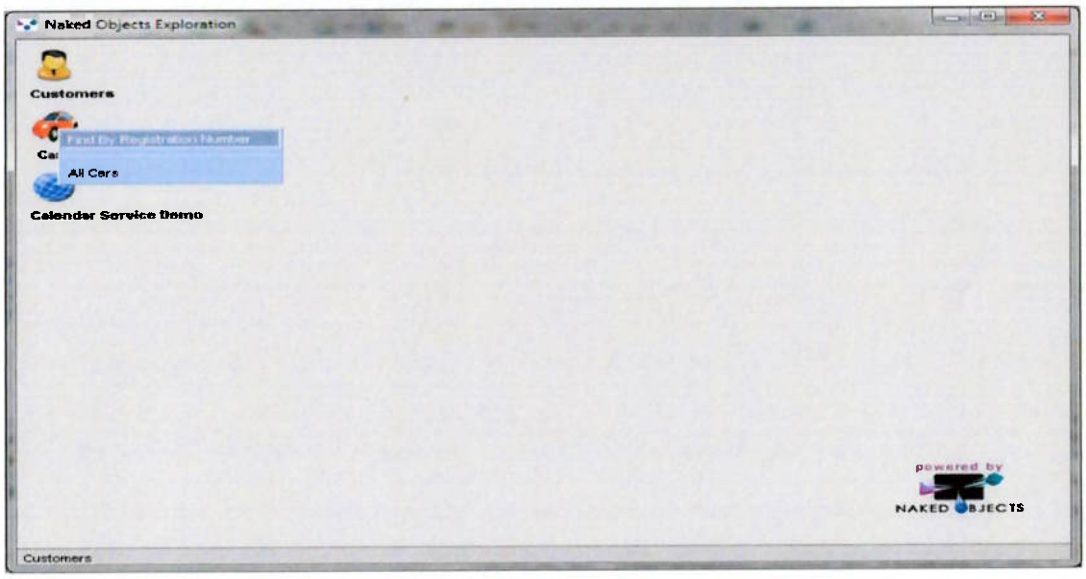
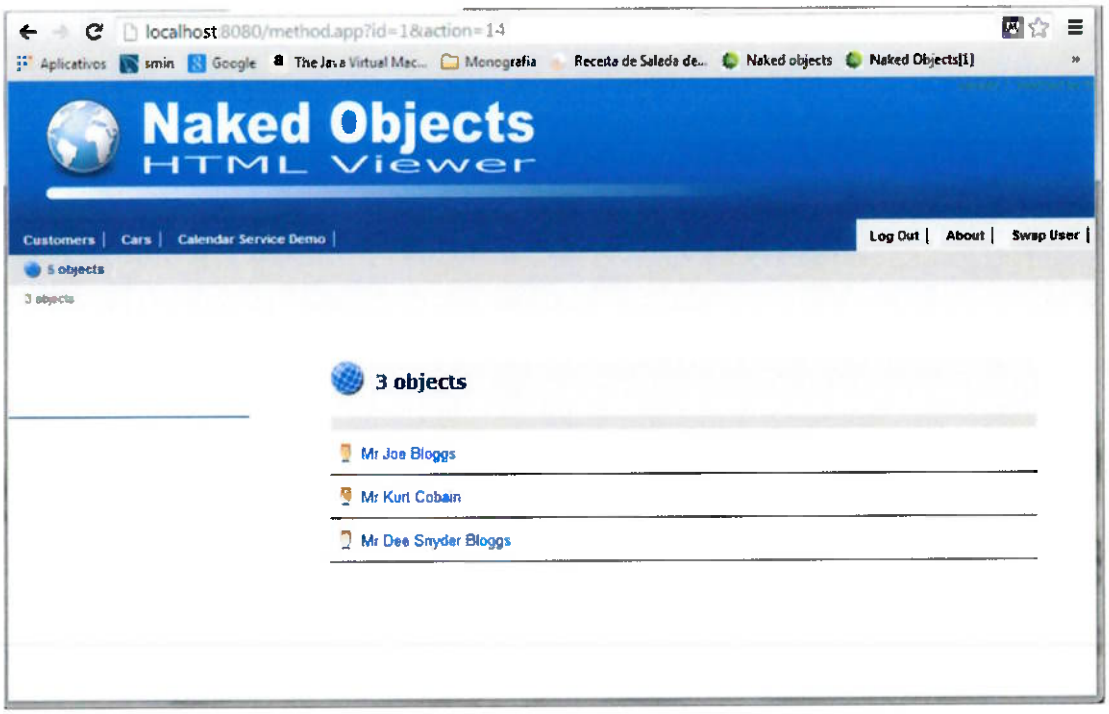


Figura 2 - Tela do framework Naked Objects aplicado ao CarServ em interface HTML



A figura 1, apresenta como ficaria a aplicação desktop client-server, gerada automaticamente pelo framework Naked Objects. A figura 2, seria a aplicação gerada em HTML, no formato de aplicação Web. Este segunda geração pode ser considerada como o embrião do Apache Isis.

Haywood explora o framework através de conceitos do Domain Driven Design, proposto por Evans (2004), tendo como proposta principal alcançar uma OOUI ideal. Através dos anos, o Naked Objects evoluiu para dois projetos

diferenciados apenas por tecnologia de implementação final, sendo o Apache Isis utilizando a tecnologia Java sob a responsabilidade de Haywood, e o Naked Objects utilizando o Microsoft.Net, mantido por Pawson e Matthews. O Apache Isis foi reconhecido e promovido como um projeto oficial Apache em meados de Outubro de 2012 (HAYWOOD, 2012).

Atualmente, o Isis incorpora uma série de frameworks e produtos open-source específicos do mercado voltado para a tecnologia Java, que visam tornar o produto final uma solução completa em termos de definição de arquitetura de sistemas. Oficialmente, se utiliza a tecnologia Java, especificamente voltada para a Web, implementando alguns frameworks conhecidos como o Apache Wicket (camada de View), implementações das especificações de mapeamento objeto-relacional JPA/JDO (Java Persistence API/Java Data Objects) através do DataNucleus, implementação de ferramenta de geração de log Apache Log4J, a ferramenta de segurança e controle de acesso Apache Shiro, entre outros.

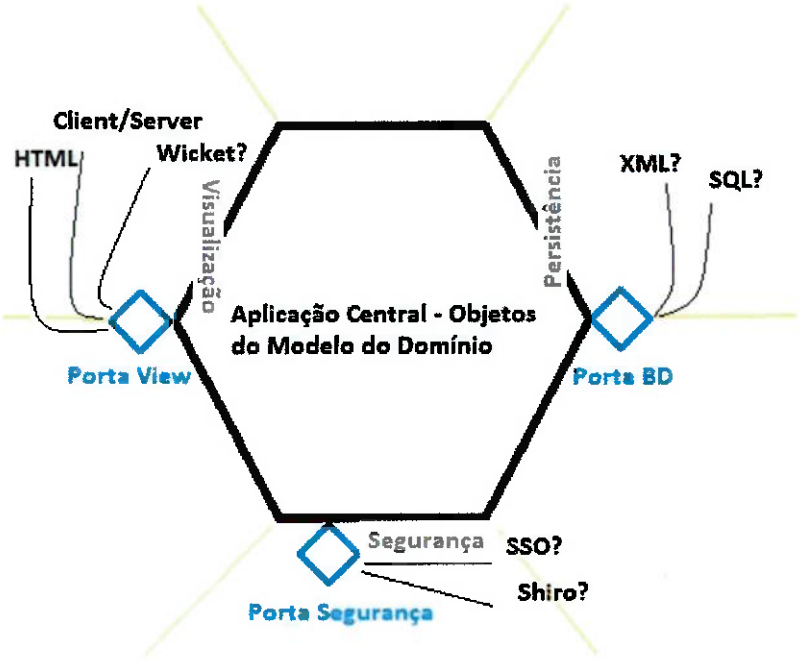
Está em um estágio de constante evolução, com uma comunidade de colaboradores ativa, liderada pelo próprio Haywood. Como parte de conhecer melhor a ferramenta, o autor deste trabalho participou como visualizador, do fórum de gestão e desenvolvimento do Isis, para desenvolvedores e usuários. Nesta lista, diversos usuários contribuem com suas experiências e dúvidas ao utilizar o framework, e através de situações inesperadas de erros ou de sucesso, são trocadas informações valiosas para esta citada evolução.

O Apache Ísis tem como seu principal objetivo, implementar os objetos definidos na identificação do modelo de domínio, fazendo com que seu utilizador se preocupe com este item apenas, deixando de lado decisões arquiteturais adicionais das outras camadas da aplicação. Outros benefícios defendidos pela comunidade do Isis são a facilidade no processo de prototipoção e na construção da aplicação quanto a quesitos de rapidez e baixa complexidade. Por ser um projeto de uso livre, também dá suporte para customização e extensão, como a alteração para a utilização de outras interfaces. Outra característica importante do Isis é a adoção de um modelo de arquitetura hexagonal (COCKBURN, 2005).

O objetivo desta arquitetura é prover pontos de conexão com diversos tipos diferentes de "adaptadores", que podem ser desde uma interface HTML de entrada, como uma saída de dados por um arquivo de texto, ou uma persistência de dados. Utilizando essa arquitetura, o modelo de domínio permanece preservado e com sua

riqueza de expressão do negócio. O formato de hexágono apenas foi escolhido pelo autor para facilitar na confecção do diagrama ao se encaixar novos adaptadores ou portas. Uma ilustração de uma arquitetura hexagonal pode ser visualizada na figura 3.

Figura 3 - Exemplo simplificado da arquitetura hexagonal com algumas portas "ativas"



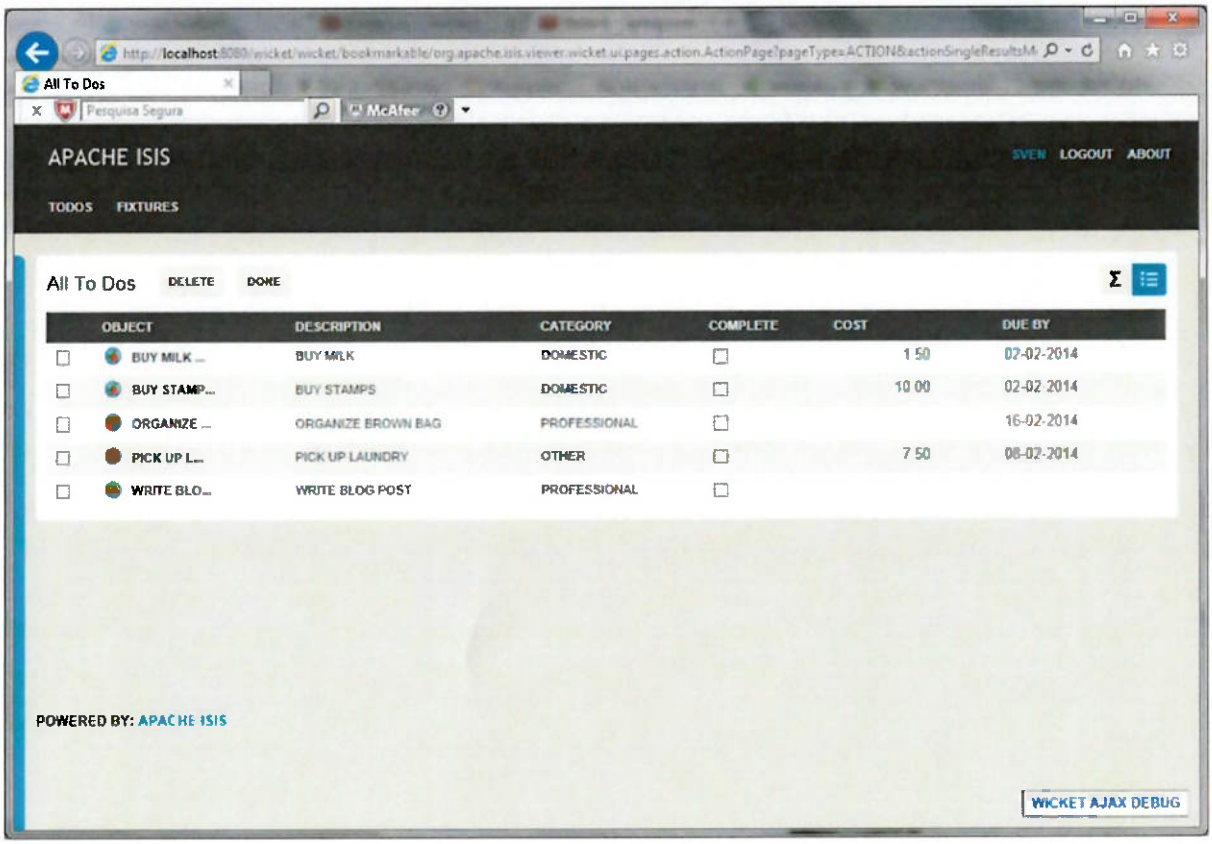
Ao focar prioritariamente nos objetos do domínio, o utilizador do framework deve se preocupar com decisões menores de arquitetura do Isis, para que o mapeamento seja feito corretamente dependendo da solução escolhida.

2.4 Especificaçao do Apache Isis

O framework funciona através de um mapeamento direto dos objetos, o que significa que a escrita do código Java pelo desenvolvedor deve ser feita da forma padrão consolidada, ou seja, através de POJOs (Plain Old Java Objects), que é uma nomenclatura criada para descrever objetos de estrutura simplificada, sem convenções impostas por frameworks que alteram a escrita do código fonte. Este tipo de classe, levando em consideração a aplicação dos padrões Entidades e Objetos de Valor, por exemplo, serão mapeados em tela.

Na figura 4 é apresentada uma tela de um sistema padrão disponível na página do Apache Isis, com uma aplicação de exemplo para gerenciamento de tarefas (*TodoItem* como na expressão em inglês *To do* - tarefa):

Figura 4 - Tela de listagem mostrando a lista de To Dos(Tarefas) de forma direta



Na figura 4, podemos ver dois itens no menu. O primeiro item de menu chamado *To dos* representa o principal modelo de exemplo usado na distribuição da ferramenta Apache Isis, implementado como modelo padrão a ser seguido. O segundo é uma estrutura de Fixtures, que o framework traz como opção para injetar dados através de objetos de teste codificados pelo desenvolvedor, a fim de simular o comportamento real da aplicação, sem se preocupar ainda com qualquer utilização de uma camada de dados.

Além disso, a ferramenta faz extenso uso de *annotations* em Java, recurso criado a partir da versão 1.5, para que seja possível automatizar tarefas através de metadados com palavras chave, iniciados por "@". No caso do Isis, é possível usar diversas *annotations* próprias (@MemberOrder, @Hidden, @Named, @Optional, entre diversas outras). No caso do @MemberOrder, a utilização indica ao framework em que ordem o item que está sendo especificado no código aparecerá na tela, levando em conta a ordem crescente de números inteiros maiores que zero.

De modo geral, a ferramenta Isis efetua seu mapeamento entre objetos e estruturas de tela ao identificar qual o tipo do objeto, e isso é feito principalmente por

algumas classes especialistas próprias da API. Um exemplo é a classe abstrata do framework, chamada AbstractFactoryAndRepository, que é a classe responsável por prover funcionalidades dos padrões de Fábrica e de Repositório para classes que herdem ela, e uma das principais do framework.

Para este tipo de objeto específico, o Isis considera que este deve ser um item visível de menu, caso o desenvolvedor não identifique o contrário. Isto significa, que qualquer classe criada pelo desenvolvedor que herda a classe AbstractFactoryAndRepository, será aplicada como um item de menu. Este tipo é uma peculiaridade do framework, e que serve como parametrizador para que seja feita a transformação do objeto desta classe em item renderizado em tela. Uma *annotation* disponível para este tipo de classe é a @Named. Ao verificar as figuras anteriores, é possível verificar que o menu corresponde ao seguinte trecho de código das classes que já vem pré-implementadas:

Figura 5 - ToDoItems herda da classe AbstractFactoryAndRepository

```
@Named("ToDos")
public class ToDoItems extends AbstractFactoryAndRepository {

    // {{ Id, iconName
    @Override
    public String getId() {
        return "toDoItems";
    }

    public String iconName() {
        return "ToDoItem";
    }
    // }}
```

Neste caso, esta classe *ToDoItems*, será a responsável por fornecer todas as ações disponíveis para instâncias de ToDoItem, e como esta poderá ser manipulada. Neste código podemos localizar a annotation @Named("ToDos"), que é a instrução utilizada para dizer ao Isis que este será o nome do menu a ser adotado ao renderizar as funções desta classe Fábrica/Repositório.

Para cada método implementado nesta classe, o Isis irá fornecer um item de menu, como visualizado na figura a seguir:

Figura 6 - Menu do Isis trazendo as ações possíveis, que estão implementadas na classe ToDoItems AbstractFactoryAndRepository


```
// {{ newToDo  (action)
@MemberOrder(sequence = "3")
public ToDoItem newToDo(
        @RegEx(validation = "\\w[@&:\\-\\,\\.\\+ \\w]*") // words,
        @Named("Description") String description,
        @Named("Category") Category category,
        @Optional
        @Named("Due by") LocalDate dueBy,
        @Optional
        @Named("Cost") BigDecimal cost) {
    final String ownedBy = currentUserName();
    return newToDo(description, category, ownedBy, dueBy, cost);
}

// {{ allToDos (action)
@ActionSemantics(Of.SAFE)
@MemberOrder(sequence = "4")
public List<ToDoItem> allToDos() {
    return allToDos(NotifyUserIfNone.YES);
}
```

Na prática, isto significa que temos quatro métodos implementados em nossa classe de Repositório/Fábrica: NotYetComplete, Complete, NewToDo e AllToDos.

Ao ser acionada a função de incluir nova tarefa(New To Do), o Isis irá renderizar cada atributo e cada método, como campos ou como botões já dentro da tela de edição ou criação de nova tarefa. Este objetivo é alcançado ao se codificar os itens que serão campos como parâmetros de entrada para este método. Abaixo, as figuras 7 e 8 apresentam as telas de demonstração de uma inclusão ou de edição para este modelo padrão:

Figura 7 - Tela de cadastro de uma nova tarefa ("ToDo")

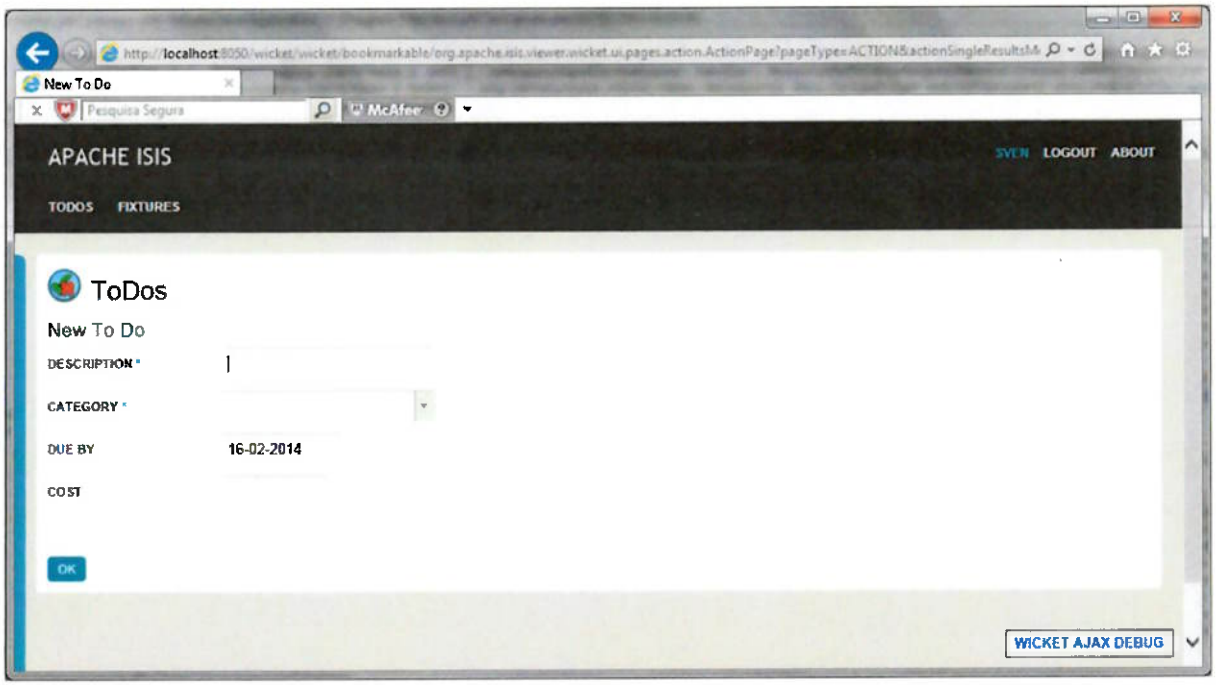
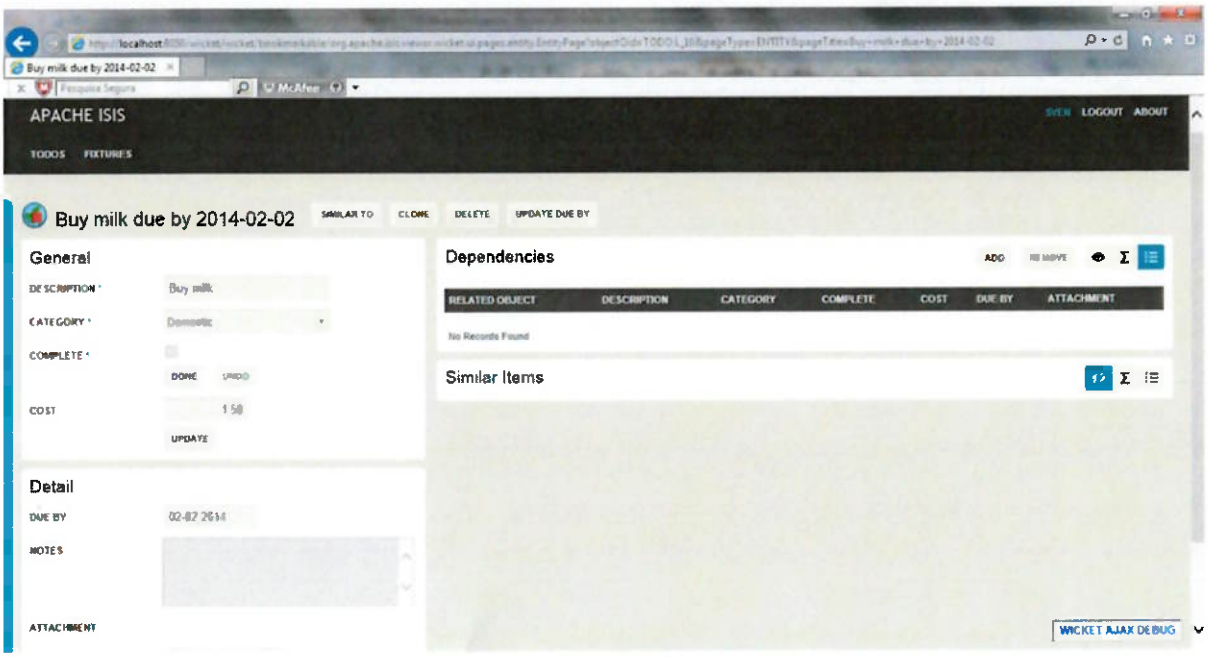


Figura 8 - Tela de detalhe de uma tarefa ao se clicar em algum item na listagem da primeira imagem



Quando um objeto é escolhido em uma tela, como em uma listagem no formato de tabela, o mesmo é mapeado por completo. O Isis gera campos para as propriedades, e botões para as ações. Como mostrado na figura 8, um To Do possui as seguintes propriedades: Descrição, Categoria, entre outras. E também possui métodos que irão prover ações através de botões como Update (Atualizar). Interessante também verificar na figura que possíveis dependências, como uma propriedade que é uma coleção de objetos, é renderizada como uma tabela por padrão localizada a direita da tela.

3. Proposta de Evolução de Software

Este capítulo apresenta o cenário atual do software Logístico, o processo trancorrido para a confecção do mesmo e a proposta de evolução, através da utilização da ferramenta em estudo, o Apache Isis.

3.1 Apresentação das deficiências do software atual da transportadora

A principal proposta para o trabalho é a utilização do framework Apache Isis em conjunto com o DDD, através de um exemplo real, para dar foco a dois itens importantes a serem avaliados e estudados: melhorias nos processos de especificação e rapidez na geração do software, focando na fase de prototipação.

O software atual é detalhado no capítulo 4, no estudo de caso que irá implementar os itens propostos. Porém, em linhas gerais, foi desenvolvido utilizando uma tecnologia não orientada a objetos, e através de um levantamento simples de requisitos com o usuário. A especificação foi feita com fluxogramas simples, e com coleta de informações principais das telas, como dados necessários e design desejado das telas através de rascunhos feitos a mão.

A prototipação inicial foi feita em HTML, o que demandou um trabalho inicial de desenvolvimento. Qualquer alteração nas demonstrações para os usuários responsáveis são necessárias novas alterações no código. Foi necessário um esforço de desenvolvimento considerável para construir o protótipo HTML, além do comportamento das telas em Javascript, uma vez que não foram utilizados frameworks de mercado para interação com as telas.

Ao longo do tempo de construção do software, incluindo fases de construção e testes, houve alguns problemas que se tornaram visíveis com o tempo:

- As telas que não eram cadastro básico (Motoristas, Veículos) e, portanto necessitavam de muitas ações, se tornaram grandes e complicadas para construção e manutenção ao longo do tempo. É o caso da tela de Relatório de Viagens;

- A falta de uma tecnologia orientada a objetos, fazia com que estas telas com muitos detalhes e ações demandassem relacionamentos de maior complexidade no modelo de banco de dados, e desta forma causou manutenções constantes nas principais entidades que acessavam estes dados;

- Não utilizar um processo formal ou alguma técnica de análise e levantamento levou a problemas na especificação das telas, assim como no

entendimento, o que gerou constante necessidade de manutenção nas mesmas telas e entidades;

- À medida que o Software agregava funções, a arquitetura adotada se mostrava ineficiente de forma geral;

O que se tem ao final deste processo é um software com uma produtividade comprometida, devido às escolhas que foram feitas na utilização de técnicas e ferramentas não produtivas, e de certo modo consideradas ultrapassadas. Por isso, existe a dificuldade na manutenção e na evolução do software, não agregando um fluxo de trabalho atualizado, em sintonia com as práticas adotadas hoje em dia no mercado de TI para a construção de software.

3.2 Domain Driven Design e Isis

Visando um estudo que traga melhoria substancial nas partes falhas identificadas, foram avaliadas técnicas atuais que estão em evidência na utilização em projetos de desenvolvimento. A utilização de uma abordagem orientada a objetos para modelagem do processo foi um dos principais objetivos a serem buscados, principalmente pelos benefícios já conhecidos neste tipo de paradigma.

Além do paradigma, busca-se uma utilização do mesmo para que se possa trazer cenários favoráveis além da representação do modelo. Existem hoje, muitos aplicativos espalhados por grandes empresas que apesar de construídos em tecnologias modernas como Java e .Net, possuem modelagens deficientes em termos de design e aderência com os requisitos, ou até complexas em situações desnecessárias. Um exemplo disso é o uso exagerado de herança, ou padrões de projeto que têm complexidade de utilização grande. Este tipo de situação pode elevar a dificuldade no momento de uma manutenção, ou até de uma refatoração do código. Isto pode levar a um maior custo no processo de manutenção.

Portanto, o DDD se torna uma altenativa interessante, que traz para o projeto uma base de conhecimento que foi construída através da junção do trabalho de muitos anos de experiências de fracasso e de sucesso na implementação de software, e desta forma agrega maturidade de conceitos.

Não só o DDD, mas a utilização de um framework também facilita este processo de evolução. Atualmente, existem inúmeros frameworks que visam atender

os mais diversos problemas que podem surgir em qualquer uma das fases de um projeto de software.

O Apache Isis, através do paradigma do Naked Objects, propõe como objetivo de sua implementação, gerar aplicações rápidas, modeladas com uma técnica como o DDD, e que utilizamuma arquitetura como a hexagonal, extensível e dinâmica, trazendo inúmeras possibilidades de configuração dependendo da necessidade. Mas principalmente, isola a parte mais importante do software, que é o modelo de domínio, ou seja, os objetos que importam para os usuários e que atendem as especificações recomendadas pelo DDD.

3.3 Enriquecimento do modelo de negócio

Um dos principais pontos negativos que este projeto como todo apresentou foi a falta de utilização de algum método de levantamento e especificação dos requisitos do sistema a ser construído.

Para efetuar uma nova modelagem de negócio, ainda mais para utilizar uma linguagem orientada a objetos, o DDD surge como uma alternativa que pode trazer alguns benefícios, como:

- Modelo com riqueza de detalhes e representação fiel das situações do dia-a-dia, transformando o software em uma extensão do trabalho executado;

- Adoção de Design Patterns e de uma DSL, que auxiliam na criação de um código conciso e expressivo quanto ao negócio;

- Criação de uma linguagem única e uma cultura rica acima do processo que está sendo representado no modelo;

Um diagrama de classes é uma ferramenta interessante a ser utilizada, podendo possuir o máximo de informação possível para nortear o desenvolvimento e representar o negócio na notação de objetos de forma natural. Evans (2004) adota o diagrama de classes como principal integrante para estes passos, mesmo não especificando o mesmo como obrigatório.

3.4 Framework orientado a objetos compatível com o DDD

Uma vez que o processo de análise e especificação tenha gerado um produto como um diagrama de classes hipotético, é possível começar a pensar em detalhes

da implementação. Normalmente, existe uma interdependência dos processos de levantamento de requisitos e confecção de protótipo. O protótipo costuma ser um dos primeiros itens a serem feitos após a identificação dos principais requisitos do software, mas nem sempre esta ação é feita programaticamente, sendo comum o design via criação de imagem representando as telas, ou utilizando aplicativos como o Power Point presente no pacote Office da Microsoft.

Mesmo não programando uma interface navegável para o cliente, um protótipo como estes em HTML, ou até em imagem, normalmente trazem apenas uma evolução de como a tela irá ficar. Agregam pouco em termos de entendimento do negócio, e representam uma camada fina do software que ainda deverá ser construído, focando demais na representação dos dados e até mesmo validação de design. Este processo de prototipação pode levar um tempo considerável, e desta forma não agregar nada além de uma geração de interface para aceitação do usuário.

Ao se utilizar uma ferramenta como o Apache Isis é possível programar as classes iniciais do software, que já estariam definidas após o processo de confecção do modelo de negócio, e obter um resultado rápido em termos de prototipação. Desta forma, o processo de análise e definição do modelo viria obrigatoriamente antes da prototipação, uma vez que o framework precisa desta definição para a geração da interface. Como proposta, o DDD pode ser adotado para a definição deste modelo. Não é necessário efetuar todas as iterações da análise para iniciar um protótipo navegável para o cliente, mas sim as principais entidades a serem utilizadas. Desta forma, ganha-se dinamismo, e maior interesse do usuário na participação do processo.

Este protótipo gerado seria uma representação do trabalho feito ao se aplicar o DDD, e quanto maior a fidelidade em termos de detalhe, maior a fidelização do que será apresentado para o usuário em termos de tela, ou seja, através de um processo ágil, é possível atacar duas frentes no ciclo do desenvolvimento.

A parte interessante desta aplicação, é que em nenhum momento os desenvolvedores precisam codificar itens de tela. O Isis tem por padrão a utilização do Apache Wicket, que é um framework de especificação de telas, e que possui um portfólio de componentes amigáveis no mesmo padrão encontrado em aplicações feitas na Web. O Isis irá efetuar o mapeamento, e suas APIs internas serão

responsáveis pela geração desta interface, deixando uma transparência entre modelo de classes e a apresentação destes como itens de tela.

Com isso, é possível evitar abismos entre as especificações de negócio, as especificações técnicas de desenvolvimento e o código desenvolvido. É comum haverem níveis diferentes de documentação nos projetos de desenvolvimento. Isto causa uma situação indesejável, aonde se tem diversos diagramas criados que vão cada vez mais se aprofundando desordenadamente, e desta forma aumentando a complexidade. Existe um momento em que a documentação sofre com falta de correspondência entre cada artefato, aumentando a dificuldade para atualização e evolução. Isto pode facilmente levar a uma obsolescência, transformando a documentação em um artefato desatualizado ao qual evita-se efetuar manutenção.

Evans (2004) reforça um conceito, que denomina alguns modelos de software como anêmicos. Um modelo anêmico é o modelo que tem em sua representação, um conjunto de objetos sem nenhuma ou pouca responsabilidade, ou ações. São objetos que foram identificados em fase de levantamento como em um caso de uso, porém sem terem a responsabilidade de executar nenhuma ação. No final, objetos como esses acabam se tornando artefatos com muitas propriedades, servindo apenas como portadores de dados.

A situação contrária também é verdadeira. Objetos que acumulam responsabilidades demais, e desta forma diminuem drasticamente o nível de coesão do modelo. Baixa coesão é um problema constante em modelos que tentam resolver muitos objetivos com poucas entidades.

Estes dois problemas apresentados, estão entre muitos que podem levar um software a ser criado e evoluído de forma prejudicial ao longo do tempo. Espera-se neste trabalho, estudar as técnicas capazes de minimizar situações como essa, e propor formas de se agregar um maior nível de qualidade para o produto final.

Com a dinâmica apresentada pelo Apache Isis, um objeto que esteja criado de forma balanceada com uma distribuição equivalente entre dados e ações, irá servir de um subsídio importante para a criação de uma tela completa em termo de campos a serem preenchidos (dados do objeto), e possíveis botões ou componentes acionáveis (ações/métodos).

4. Avaliação dos resultados

4.1 Avaliação do Naked Objects

Com o objetivo de auxiliar na definição do Naked Objects, Pawson (2004) executou três estudos de caso em sua tese de Doutorado. Um deles foi através da implementação de novos sistemas dentro da instituição DSFA (Department of Social And Family Affairs) um departamento do governo da Irlanda, que tem como responsabilidade administrar serviços sociais. Outro estudo, foi a execução de dois projetos que utilizaram o Naked Objects para a empresa Safeway, que atua no segmento de varejo, sendo a quarta maior rede de supermercados no Reino Unido. Nestes dois primeiros estudos, o objetivo foi a renovação de sistemas legado que foram construídos em tecnologias e linguagens procedurais. E por último, a utilização de uma aplicação chamada CarServ, proposta por Haywood e Carmichael (2002), que gerencia uma prestadora de serviços de mecânica para automóveis em geral.

Estes estudos de caso foram utilizados para demonstrar diferentes pontos a favor da utilização do Naked Objects. Tanto no caso da DSFA, como na Safeway o foco está na avaliação do framework de uma perspectiva qualitativa. Para isto, o autor aplicou diversos questionários aos participantes com cargos diferenciados que participaram de todo o fluxo, desde a identificação dos requisitos, até a prototipação ou construção do sofware. Estes questionários vieram a comprovar que a aceitação do conceito foi positiva e trouxe benefícios para a empresa que estava aplicando, bem como para o processo de desenvolvimento e para o software em si.

Em ambos os casos, os resultados tiveram como consenso geral avaliações positivas. Os principais itens que foram classificados como tal para o estudo da DSFA, foram:

- Forte evidência de agilidade de utilização/operação;
- Alguma evidência de agilidade estratégica melhorada;
- Alguma evidência de melhoria da comunicação entre usuários e desenvolvedores, mas um sentimento de maior e melhor prototipação;
- Nenhuma evidência de um ciclo de desenvolvimento rápido da aplicação como um todo, porém isso pode ser atribuído ao fato de em paralelo ter ocorrido a

tarefa de desenvolvimento de um framework que aplicou o conceito Naked Objects;

- Ciclo de desenvolvimento rápido durante a fase de prototipação;
- Comunicação aprimorada entre os usuários e os desenvolvedores;
- O período de exploração, quando usado com o Naked Objects e um framework de implementação do conceito apropriado, pode ser altamente efetivo.

Para o último case, Pawson (2004) é direto ao tentar comprovar quantitativamente a aplicação do Naked Objects, levando em consideração as seguintes características constantes e uma característica variável na aplicação do estudo:

- Variável: Alteração do paradigma, projeto inicialmente feito em camadas MVC, e na segunda versão utilizando o framework Naked Objects;
- Constantes: Mesmo modelo de negócio inicial e requisitos, mesmo autor do modelo e responsável pela programação das duas versões do sistema. Para este caso, Dan Haywood na época ajudou a conduzir este estudo.

Através de iterações de desenvolvimento, e análise comparativa do resultado das duas aplicações, foram avaliados os resultados do produto final. Como resultado, temos as seguintes tabelas extraídas da tese de Pawson (2004):

Tabela 1 - Tabela retirada da seção 7.4 da tese de Pawson(2004)

	Número de Classes	Número de Métodos	Média de Métodos por Classe	Linhas de Código Java (LOC - *Lines of Code*)	Média de Linhas por Método
CarServ1	190	788	4.1	7304	9.3
CarServ2	27	230	8.5	1726	7.5

Fonte: Pawson (2004)

Tabela 2 - Comparação dos índices acerca de classes externas para cada estudo de caso

	Classes externas invocadas dentro do código da aplicação, ignorando classes java.lang e java.util	Métodos unicos em classes externas invocados, ignorando java.lang e java.util
CarServ1	142	411
CarServ2	18	56

Fonte: Pawson (2004)

Ao analisar as tabelas acima colocadas, pode-se perceber que houve uma drástica diferença entre valores. CarServ1 diz respeito à primeira aplicação, desenvolvida por Haywood (2002), e refeita para o estudo. CarServ2 diz respeito a segunda aplicação, utilizando o paradigma e o framework Naked Objects, também por Haywood (PAWSON, 2004).

A primeira tabela traz a diferença entre o número de classes, o número de métodos, a média de métodos por classes, total de linhas de código Java e a média de linhas por métodos. A segunda tabela traz o número de classes externas utilizadas no código da aplicação e os métodos de classes externas chamados, ignorando chamadas de bibliotecas Java padrão, como Java.lang e Java.util.

Nos dois casos, foi possível verificar como o Naked Objects transforma as características do projeto, trazendo menores índices interessantes, como o menor número de classes, métodos e linhas de código, média de linhas por método(Tabela 1), assim como menor número de classes externas e métodos de classes externas1(Tabela 2). Estes índices, podem significar uma redução de esforço de construção e manutenção, e até de custo na implantação de projetos de software. O ponto interessante, é destacar que um processo de construção ou manutenção poderia ser beneficiado por esses resultados, com a simplificação do modelo especificado e implementado.

4.2 Aplicação da proposta para o software de controle logístico atual não Orientado a Objetos – Apache Isis em ação

Com o objetivo de aplicar os conhecimentos adquiridos via leitura do principal material, será apresentado uma aplicação prática do Apache Isis. Para esta aplicação, é utilizada o software já existente, no qual o autor desta monografia atuou como analista e programador, e rapidamente referenciado no capítulo 3.

Trata-se de um projeto para uma empresa de pequeno porte de transportes, que necessitou de um sistema que controlasse as viagens que os motoristas faziam de forma a informatizar um processo feito manualmente. Este processo era feito através de anotação em uma lista impressa de tempos em tempos, bem como também era manual o gerenciamento de cadastros de Motoristas, Veículos, entre outros.

Inicialmente, este sistema tinha como objetivo final, fornecer um sistema completo que contemplasse todo o ciclo do transporte efetuado, ou seja, cobrindo desde o registro da demanda, o atendimento do motorista, o fretamento do serviço a ser prestado, o controle financeiro do mesmo com integração de caixa, a geração de relatórios e conhecimentos de transportes a serem impressos e emitidos aos clientes.

Devido a alguns entraves, o projeto não evoluiu tanto quanto deveria. O principal motivo foi a complexidade a ser traduzida em software, integrando diversos papéis de executores dentro da empresa, considerando também o trabalho de reformulação de todo processo executado. A aplicação não poderia desviar do principal objetivo de exercer, exatamente, o mesmo tipo de trabalho que uma planilha Excel personalizada pelo usuário poderia fazer. Os usuários dependiam intensamente dessas planilhas, utilizando de forma extensa os recursos da ferramenta Office. Porém, ainda sim era importante a construção de software, trazendo uma unificação de conhecimento, padronização de entrada de dados, além da centralização e segurança da informação. Entre os usuários que seriam beneficiados com o sistema, estavam um analista financeiro, um operador logístico e o dono da empresa, que faz o papel de coordenador logístico.

Devido à criticidade do modelo de atendimento ofertado, e à estrutura enxuta do departamento que controlava esse processo, um sistema muito avançado deveria ser desenhado. Isto causaria aumento no custo do projeto para a empresa, e o consultor não poderia ofertar sem uma equipe maior de desenvolvimento. Ambas as

partes decidiram deixar parte do processo com o usuário como estava, apenas complementando com a funcionalidade principal de controle de viagens, o que simplificou o projeto.

O software atual de controle logístico foi desenvolvido em linguagem procedural, dentro de uma arquitetura simples de aplicações web com trocas de informação entre um servidor de aplicação Microsoft IIS (Internet Information Services) e um banco de dados de uso livre e popular, MySQL, além de uma interface HTML acessada via browser pelos usuários. A tecnologia escolhida foi o Microsoft ASP (Active Server Pages), juntamente com utilização de Javascript para manipulação desta interface e requisições assíncronas via AJAX (Asynchronous Javascript and XML) .

A especificação da aplicação pode ser resumida da seguinte forma:

Um sistema que deve controlar as viagens feitas pelos motoristas funcionários ou terceiros, que vão executar um serviço de transporte de cargas para clientes externos. Esta viagem, no momento da execução deve apresentar data de saída, bem como o nome do motorista que a está executando, qual veículo irá utilizar (em caso de terceiro, o veículo particular), origem/destino da viagem, total de frete a ser cobrado em caso de ser motorista prestador de serviço, adiantamento dado ao motorista referente ao pagamento e saldo a pagar. Outras informações secundárias serão necessárias, como número do conhecimento que será atualizado posteriormentes, número de Página Branca (relatório interno que descreve a viagem) e informações como responsável no cliente e centro de custo da empresa cliente.

Para que este controle funcione, é necessário o prévio cadastramento de Motoristas (funcionários e prestadores de serviço utilizam o mesmo cadastro, sendo classificados por um campo de tipo de informação), Veículos da empresa e Empresas cliente.

Seguem as figuras 9, 10 e 11 ilustrativas do sistema atual:

Figura 9 - Tela de Relatório de Viagens

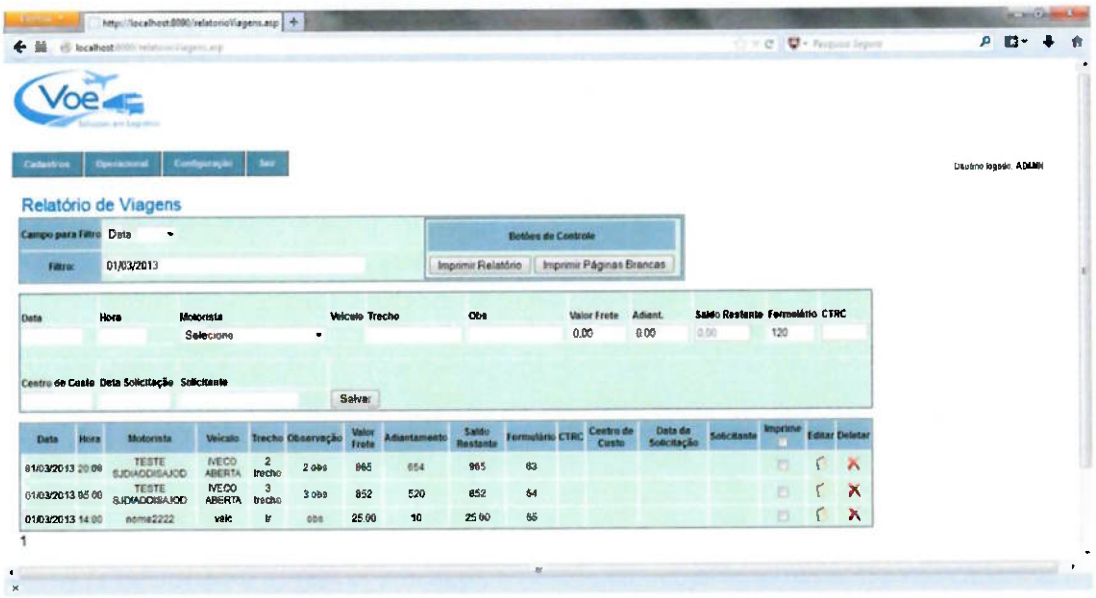


Figura 10 - Tela de listagem de Veículos

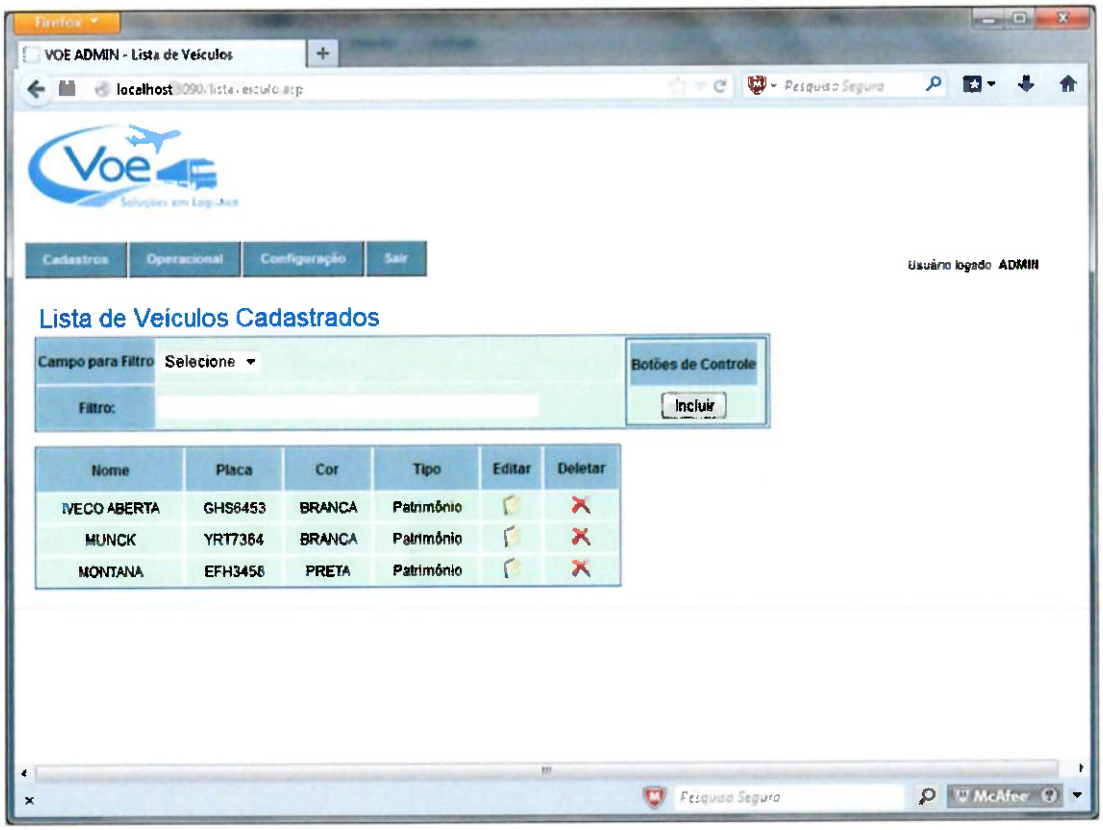
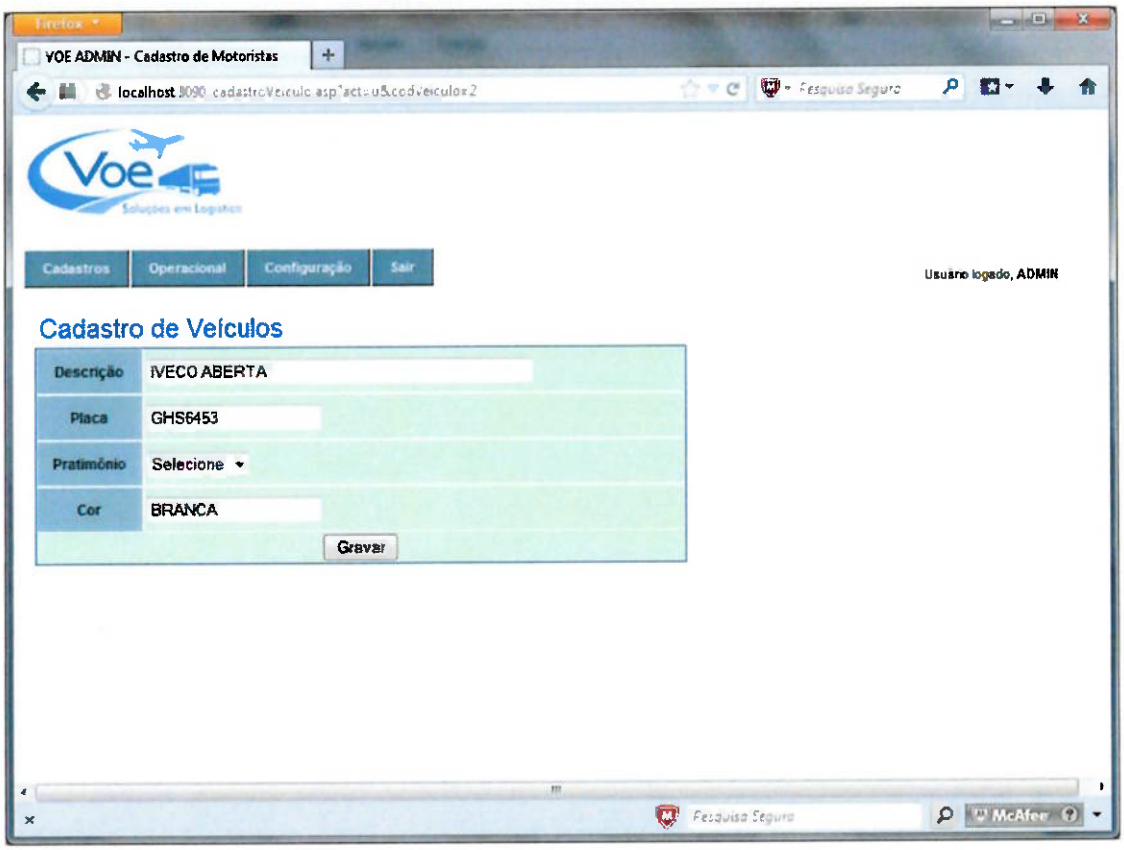


Figura 11 - Cadastro de Veículos



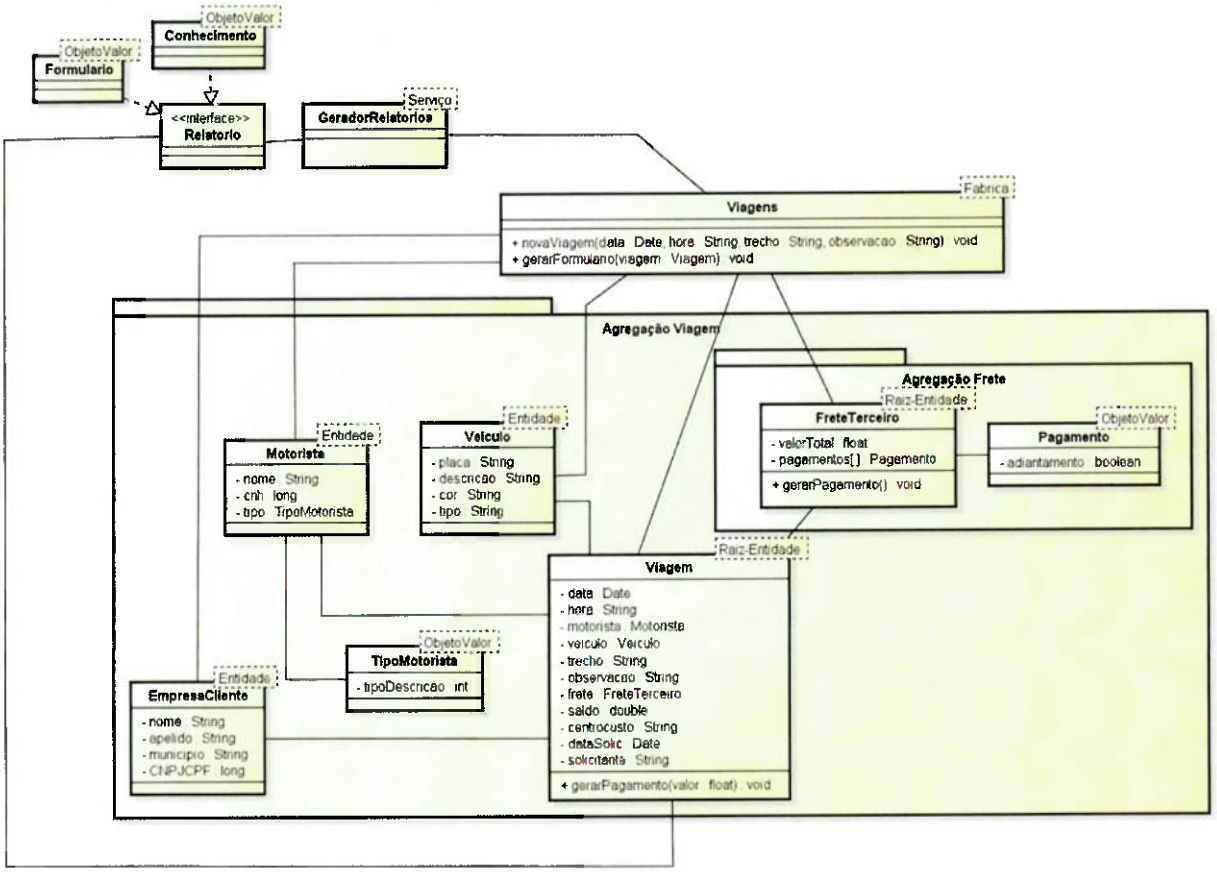
4.3 Domain Driven Design aplicado ao problema

Com o sistema atual funcionando, em processo apenas de melhorias, pode-se considerar que existe uma base de conhecimento razoável a se iniciar no processo de exploração.

Uma vez que o modelo anterior era focado em casos de uso e em tarefas, ainda sem utilização do paradigma orientado a objetos, a exploração dos conceitos de outra perspectiva se fez necessária. Desta forma, foram executadas algumas entrevistas com o usuário principal do sistema.

O modelo inicial, levando em consideração a descrição apresentada no início deste capítulo, e o levantamento que foi executado novamente, é representado na Figura 12 através um diagrama de classes.

Figura 12 - Diagrama de Classes com a aplicação do modelo de domínio



A principal classe identificada é a Viagem na figura 12. É a base para que o sistema funcione e é o principal controlador do processo da empresa para que o sistema faça o que se propõe, gerenciar as viagens. É classificada como uma entidade, pois cada viagem é única, e tem como uma identidade própria, como o número de conhecimento para mesma e o número de formulário. Outra classe interessante é a Viagens, que faz neste caso o papel de Repositório/Fábrica de Viagens.

Algumas outras classes secundárias foram criadas como Motorista e Veículo. Para este estudo, não foi desenvolvido um diagrama de classes completo. Para que isto fosse atingido, seriam necessárias inúmeras iterações com o usuário, e isto demandaria um esforço não necessário para esta prova de conceito acerca da prototipação via Apache Isis. A ideia principal é verificar a facilidade e rapidez com que o protótipo foi gerado, justamente considerando fases preliminares de especificação de um modelo de domínio.

4.4 Naked Objects/Apache Ísis aplicado ao problema

O resultado esperado para esta aplicação é um protótipo ou até aplicativo em passos iniciais que aborde de uma maneira direta o mapeamento de comportamento entre entidades e interface.

O primeiro passo foi implementar as principais classes encontradas no modelo apresentado acima: a entidade da Viagem, e seu relacionamento com o objeto do tipo Fábrica/Repositório Viagens. Neste caso, é possível fazer uma mapeamento lógico entre a Viagem e a classe ToDoItem (item tarefa) padrão do Isis. O mesmo pode ser observado entre a classe Viagens e a classe ToDoItems (itens tarefa). Desta forma, a aplicação inicial foi totalmente baseada no padrão já utilizado, criando as novas classes com os atributos e métodos corretos, previstos no modelo.

Além disso, foi também feita uma implementação da classe Veículo e como ficaria a tela de edição e inclusão de uma nova instância. Para este desenvolvimento como citado acima, foram usadas as classes modelo do framework. No caso, como foram implementadas duas classes de Entidades, como a Viagem e Veículo, foram geradas funcionalidades de edição e inclusão para cada uma delas, e tela de consulta com uma listagem em formato de tabela.

Para que se tenha acesso a estes itens da lista ou funcionalidade de menu para acessá-las, foram implementadas classes do tipo AbstractFactoryandRepository. Desta forma, o protótipo apresenta dois itens de menu, um através da classe Viagens com a *annotation* @Named("Viagens"), para que o menu trouxesse esta descrição e a outra classe Veiculos, responsável pelo item de menu de mesmo nome, com a *annotation* @Named ("Veículos").

Neste pequeno processo de desenvolvimento, também foi necessária a alteração em outros itens que fazem parte do Isis, como no arquivo *isis.properties,* que contém propriedades importantes para o funcionamento da aplicação. Um exemplo é a especificação de qual o tipo de implementação para uso de persistência e consulta de dados. Existem algumas opções, como a utilização de classes em memória, ou fazendo a conexão com um banco de dados. Outra possível configuração é a especificação das *Fixtures,* classes que servem de apoio para os testes, e que são responsáveis por injetar os dados para a prototipação. No caso deste aplicação, foram utilizados objetos de Fábrica/Repositório do tipo Fixtures, que

geram objetos ou até coleções dos mesmos com dados fictícios, possibilitando a execução do protótipo com dados pré-existentes.

Para este estudo, foram necessárias algumas ferramentas importantes, como o Eclipse. O Eclipse é um software popular para o desenvolvimento de aplicações principalmente em Java, classificado como uma IDE (Integrated Development Environment). Além desta ferramenta, foi utilizado também o Maven. Este é uma ferramenta de automatização de instalação do software, que visa padronizar o projeto com suas dependências e através do mesmo, executar o comando de instalação instantânea.

O Isis possui sua implementação baseada no Maven. Neste caso, este passo de instalação é executado através de comandos do Maven apontando para o proprio site do Isis, que efetua o download de todo o código fonte através das instruções que estão pré configuradas no framework. Este passo é essencial para a instalação correta do framework no computador do executante.

Após o desenvolvimento das classes no período de 1 semana, foi possível se chegar ao primeiro protótipo que era o objetivo desta aplicação. O processo foi rápido, e com apenas alguns erros gerados pelo conhecimento inicial do autor da ferramenta como um todo, foi simples a aplicação desta iteração. O resultado da aplicação pode ser avaliado nas figuras 13, 14 e 15:

Figura 13 - Tela da listagem de viagens prototipada via Apache Isis

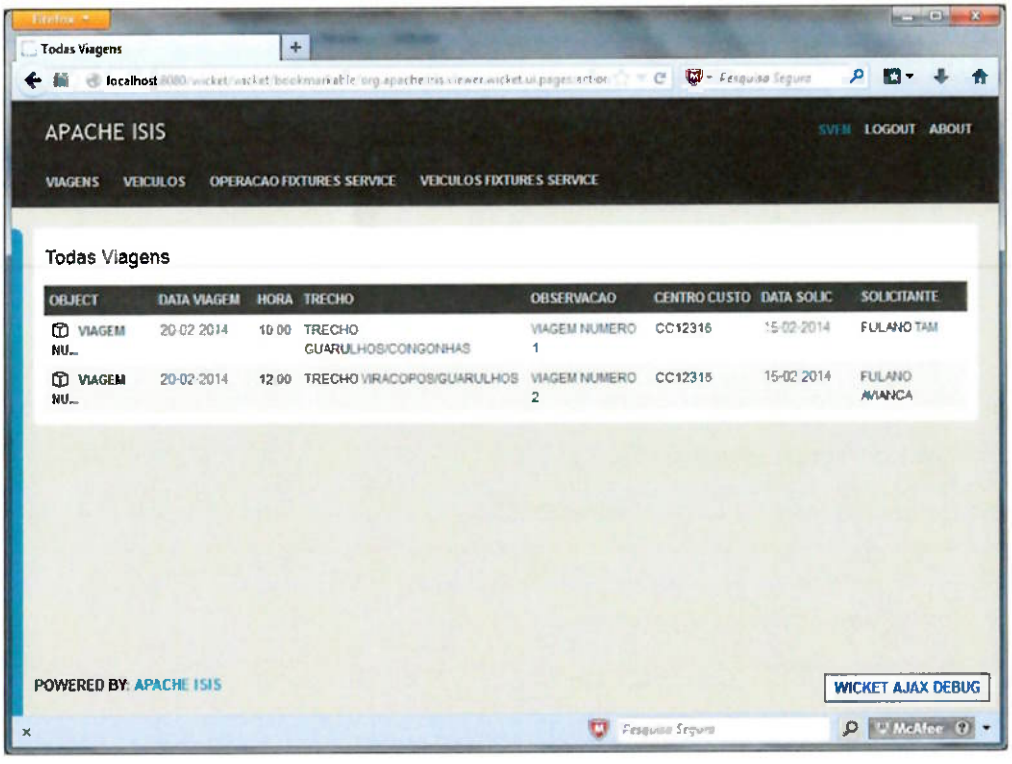


Figura 14 - Tela de Edição de uma Viagem

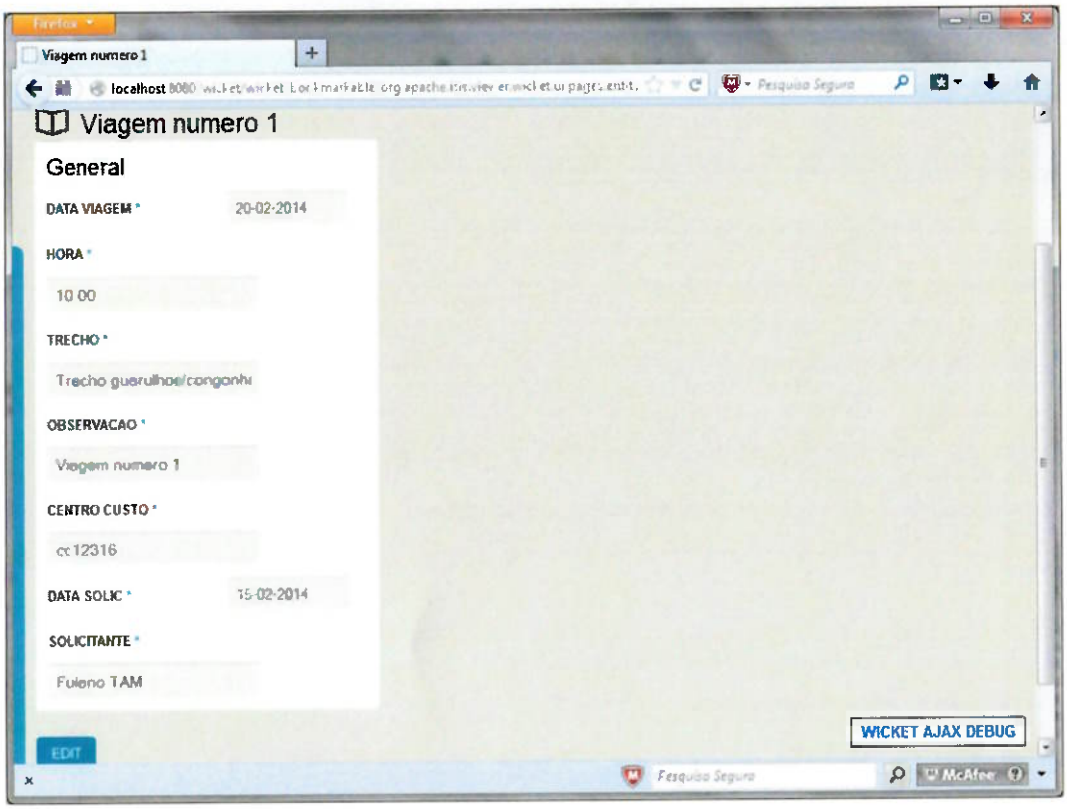
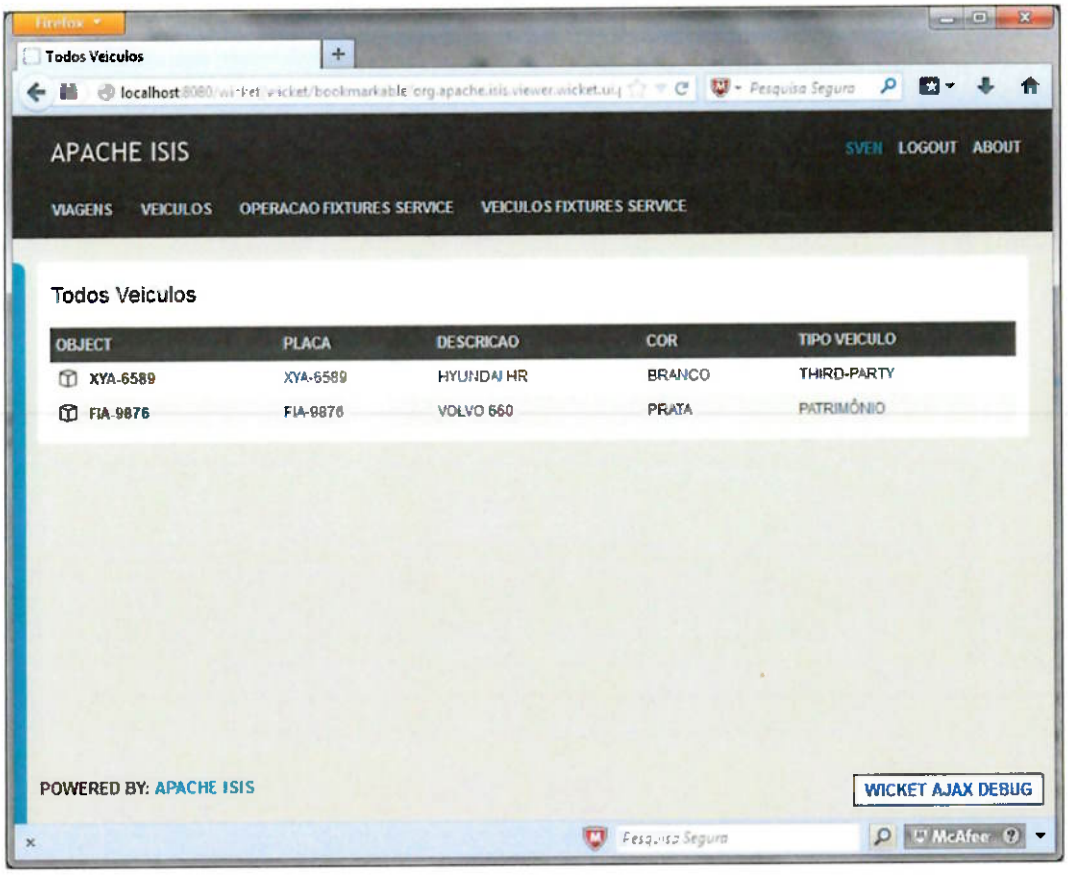


Figura 15 -Tela de listagem de Veículos



As imagens mostram a evolução das telas para uma interface amigável e com componentes com design moderno. Porém, o que deve se destacar, é a correspondência entre os objetos implementados no código e renderizados em tela.

Nas imagens 13 e 15, visualiza-se as figuras que contêm os objetos de tipos como Viagens e Veículos, sendo esse resultado motivado pela codificação de classes do tipo Fábrica e Repositório, que contém métodos que listam os objetos que serão trazidos de uma fonte de dados.

Na imagem 14, identificam-se os campos que serão utilizado para atualizar o objeto em si, neste caso uma Viagem. Ao se comparar esta imagem, com o modelo criado utilizando o DDD, pode-se verificar que existe uma correspondência direta dos dados especificados e o que foi renderizado na tela. Todos estes campos disponíveis, estão presentes como atributos da classe Viagem e são especificados através das configurações no código desta classe, através das *annotations* disponíveis para esta finalidade.

Assim, foi possível testar a geração rápida de telas, utilizando-se apenas do modelo especificado, a codificação das classes principais do domínio, como as classes Viagem(Entidade) e Viagens(Fábrica/Repositório) e o preenchimento de classes de testes para a geração de dados nos objetos em memória. Não foram necessárias codificações auxiliares em termos de interface. Isto traz a praticidade de gerar o protótipo com facilidade codificando apenas em Java.

Para um estudo completo da ferramenta, seria necessário executar várias outras iterações do projeto além da fase da prototipação, principalmente para atingir os objetivos principais dos conceitos revisados no capítulo 2.

Uma fase seguinte interessante, seria destilar o modelo o quanto fosse necessário para que o detalhamento das regras de negócio fosse se apurando com maturidade gradual, o que colocaria em prática os principais pontos do Domain Driven Design. Também seria interessante, explorar mais conceitos e formas de parametrizar a aplicação de regras no Isis, como campos que devem ficar inativos dependendo de alguma situação, ou a interação entre objetos do tipo Entidade e Objeto de Valor, ou até um Serviço.

Devido ao contexto de testar a aplicabilidade inicial do Isis, não foi possível executar maiores explorações de como o mesmo atende um modelo de negócio completo, e suas particularidades.

4.5 Avaliação dos Resultados

4.5.1 Evolução da especificação

A evolução da especificação notoriamente se mostrou de maior utilidade para possíveis implementações futuras e melhor entendimento dos processos. A representação utilizando o paradigma Orientado a Objetos, muda drasticamente os artefatos apenas interpretadores de ações, e cria um conjunto de conceitos que agregam maior valor à especificação técnica.

Consideradas as regras do DDD, foi possível identificar e classificar as classes que teriam maior correspondência com o negócio, fora a classificação utilizada entre Entidades, Objetos de Valor, entre outros. Este passo, mesmo que não tivesse precedido a aplicação do Isis, poderia, de forma interessante, fornecer um modelo para qualquer tecnologia Orientada a Objetos, como por exemplo em Java utilizando modelos arquiteturais como MVC padrão, através de frameworks de mercado.

A iteração rápida na geração de um modelo inicial também foi vantajosa. Poder utilizar regras que ajudam na definição de tipos de objetos, fez com que houvesse uma profunda reflexão para esta identificação, o que forçou um trabalho de maior relevância e atenção a detalhes ao se trabalhar com objetos.

4.5.2 Prototipação e Fase Inicial

O Apache Isis é baseado na arquitetura hexagonal, o que facilita a adaptação de seu funcionamento para diferentes implementações de um dado tipo de componente. O próprio exemplo disponível para avaliação (aplicação ToDoItem), já contém trechos de código comentados, com as diversas opções disponíveis.

Um exemplo prático é a persistência e utilização dos dados quando ainda em fase de prototipação. Uma delas pode ser feita através de gravação destes dados em memória, já outra opção seria aplicar o framework DataNucleus, que implementa um o padrão JPA/JDO. Através do DataNucleus, é possível simular uma situação mais próxima de uma possível implementação de um dado banco de dados, utilizando a persistência via mapeamento do tipo objeto relacional do JPA/JDO.

Outro item interessante, é o fato de que, utilizando o Isis, é possível exercitar a implementação das classes, mesmo que ainda em fase de evolução do modelo.

Isto faz com que seja possível uma refatoração natural de código Java e uma vez que há o mapeamento direto destes para a interação do sistema, pode-se concluir que o funcionamento e a coerência do modelo de negócio com o código implementado ganham em qualidade.

Em um processo normal de implementação de uma aplicação, indiferente sendo em Java ou ASP e indiferente de qual framework usado para camadas View e Controller, necessitaria de horas de esforço de um designer ou até mesmo de um programador front-end, para especificar e desenvolver um protótipo para validação com o cliente. Este processo poderia demorar mais do que o previsto no planejamento. Como o Apache Isis tem uma geração automática de protótipo, através apenas da codificação das classes principais, há um beneficio para o processo de análise de levantamento de requisitos. A agilidade com que foi possível implantar este protótipo estudado é um item a ser enaltecido.

4.5.3 Comparação da implementação

Ao efetuar uma comparação entre os dois processos de análise e implementação, pode-se destacar alguns aspectos importantes: diferentes formas de se explorar o problema, diferentes processos de codificação, diferenças arquiteturais entre paradigmas e tecnologias e diferentes níveis de dificuldade para novas iterações de desenvolvimento ou manutenação do código.

Explorar o problema: ao se pensar numa aplicação Web padrão como era no auge da utilização de páginas dinâmicas em ASP, era comum que se tivesse uma grande preocupação com a interface e como seria este desenvolvimento. Particularmente no ASP, força-se uma extensa e detalhada implementação de código HTML e Javascript, para a colaboração dos dados entre o ASP em si e a camada de apresentação. Ao se adotar um framework como o Apache Isis, que acumula outros frameworks como o Apache Wicket, existe uma melhoria na qualidade visual da camada de apresentação, e um aumento de produtividade ao se manipular seus componentes internos ao invés de programar cada item HTML. Por estas diferenças, explorar o problema para o Isis significa pensar em classes, para um ASP seria, como será o fluxo dos dados e como estes se apresentarão.

Processos de codificação: ao se focar numa aplicação Web como a descrita, a preocupação se divide em HTML, código ASP dinâmico, composição de

dados e como juntar estas peças. Desta forma, as regras de negócio trancendem por diversas camadas, além de se dispersarem por trechos não necessariamente coerentes para o negócio. Ao se avaliar a aplicação do Isis, o processo de codificação se limita apenas ao Java e a orientação a objetos. Outro ponto importante, é adicionar ao processo de aprendizagem, particularidades da ferramenta Isis, que acaba sendo limitada a sua comunidade de desenvolvedores, ao invés de ter fóruns consolidados e espalhados pela internet, como é o caso do ASP, ou até para Java utilizando soluções populares de mercado, como o Spring MVC.

Diferenças arquiteturais: não pode ser negado o fato que o módulo que gera uma aplicação no Isis é intrisecamente muito mais complexo que uma especificação simples de tecnologias como o ASP. Enquanto que o Isis possa ser comparado a um robô que desenvolve sob demanda baseado em parametrizações e codificação Java, uma aplicação Web simples promove uma maior simplicidade arquitetural e um maior contato entre o que o desenvolvedor pode alterar em uma aplicação em seus diversos aspectos. Um exemplo disso, é a possível manutenção em um componente HTML na tela, que rodaria em qualquer browser, contra um componente do Wicket, que implementa um código HTML pré especificado, e que aceita números limitados de parâmetros podendo diminuir a flexibilização de uso.

Dificuldades nos processos de evolução e manutenção: fica evidente, mesmo levando em consideração que a segunda aplicação do sistema em estudo era apenas um protótipo, que uma manutenção na aplicação gerada pelo Isis, requer menos tempo, menor complexidade e menor linhas de código a serem incluídas ou alteradas. Qualquer alteração na aplicação em versão ASP, custaria várias linhas de código, em diversas linguagens, e em pontos disseminados pelo sistema.

Importante também levar em consideração a diferença de idade das duas tecnologias e paradigmas.

4.5.4 Considerações do Capítulo

Apesar de não terem referências diretas entre o trabalho de Evans(2004) com o DDD e dos autores do Naked Objects, Pawson e Matthews(2002), pode-se perceber a afinidade entre as propostas de cada. Os dois convergem nas seguintes questões:

- O projeto deve iniciar através de um processo de iterações fortemente direcionadas pela participação de detentores do negócio juntamente com a equipe técnica, com uma participação valiosa desde o início visando otimizar a implementação, em questão de riqueza do modelo e benefícios na manutenção;
- Entendem que técnicas do manifesto ágil (XP Programming, Scrum, entre outros) tendem naturalmente a ser bem vindas na execução desta fase;
- Ambos defendem riqueza no modelo, que deve expressar de forma fiel o negócio, e aproveitando todo o potencial que o conceito da Orientação a Objetos pelo menos um dia mostrou ao ser materializado em linguagens de programação como Simula, Smalltalk, Java, C++, C#.
- Defendem de alguma forma a utilização do MDD (Model Driven Design), técnica que provê a geração automática de partes do software mapeadas diretamente de um modelo, como um Diagrama de Classes da UML. No caso de Evans(2004), de forma mais tímida como proposta apenas. No caso de Pawson e Matthews(2002) e Haywood(2009), substancialmente, uma vez que o Naked Objects e o Isis explicitamente utilizam desta técnica para sua geração automática;
- Um dos responsáveis pela evolução do Naked Objects em suas primeiras implementações como framework open source para o atual Apache Isis, Dan Haywood, mescla os dois conceitos, convergindo inclusive para o tema aqui executado.

5. CONSIDERAÇÕES FINAIS

5.1 Contribuições do Trabalho

Com o desenvolvimento deste trabalho, foi possível explorar uma ferramenta com uma proposta interessante, não só do ponto de vista da área de desenvolvimento, mas também de fases como a prototipação, levantamento de requisitos e análise.

A contribuição principal é fazer uma avaliação do produto atualmente ofertado como solução de uso livre, e além disso mostrar como é possível combinar conceitos poderosos envolvendo um paradigma como a Orientação a Objetos, o Domain Driven Design e o Naked Objects. Este tipo de estudo vem reforçar a utilização de uma ferramenta como essa, independente do objetivo. Mostra que de forma simplificada pode existir uma mínima contribuição para qualquer parte do processo, mesmo que o Isis seja usado para prototipar, ou até organizar uma implementação inicial de classes Java.

Para o processo de prototipação, pelo tempo que durou a implementação das novas classes no Isis, ficou claro que o processo ganha em produtividade e qualidade, podendo fornecer redução em custos para aplicação de mão de obra nesta fase e melhor visualização para os usuários em uma fase adiantada.

Para o processo de desenvolvimento, o Domain Driven Design vem sendo aplicado de forma efetiva e direcionada pelo Isis, pode-se destacar a combinação da riqueza que esta implementação do domínio pode oferecer, trazendo uma interface satisfatória para o usuário.

Através deste estudo em um contexto geral, houve uma agregação substancial de conhecimento na fase de pesquisa, leitura e conhecimento dos conceitos estudados. A abordagem do Domain Driven Design foi aprimorada, o conceito do Naked Objects conhecido mais a fundo, revelando-se um conceito poderoso e interessante do ponto de vista dos paradigmas atuais.

O Apache Isis, mostrou que é uma ferramenta que pode ser profundamente explorada e testada para avaliação em projetos reais, levando benefícios e tecnologias de ponta conduzidas por profissionais que são reconhecidos na área de tecnologia global e nos meios de pesquisa.

5.2 Trabalhos Futuros

Os trabalhos futuros possíveis para a exploração do Isis com maior profundidade são diversos. O principal deles seria continuar neste fluxo de iterações, evoluindo a aplicação até que ela pudesse servir de substituta para a versão atual. Uma especificação completa de um aplicativo teria muitos detalhes interessantes a serem analisados e testados para confrontar os benefícios defendidos pelos autores pesquisados.

Um estudo interessante seria pesquisar e conhecer mais a fundo como o Isis funciona internamente, e de que forma ele efetua este mapeamento entre objetos Java e campos e componentes de tela através do Wicket. Este tipo de pesquisa estaria focada na parte de arquitetura da ferramenta, e consequentemente demandaria um maior estudo das ferramentas Apache Wicket, DataNucleus, entre outras. Este seria o principal atrativo para o autor, despertando o interesse em conhecer como o Isis foi especificado, como foi seu projeto de arquitetura, e quem sabe conseguir atuar profissionalmente de forma oficial com o mesmo.

Por último, seria interessante também um engajamento maior no projeto, podendo no futuro fazer parte de possíveis processos de melhoria conduzidos por autores como o Dan Haywood, que é figura ativa no fórum por email do Isis, e incentiva que os participantes instalem e avaliem a ferramenta, afim de usá-la e ao mesmo tempo fazer uma espécie de teste beta.

REFERÊNCIAS

APACHE SOFTWARE FOUNDATION. Apache Isis. Disponível em: <http://isis.apache.org/>

APACHE SOFTWARE FOUNDATION. Apache Log4j. Disponível em: <http:// http://logging.apache.org/log4j/2.x/>

APACHE SOFTWARE FOUNDATION. Apache Maven. Disponível em: <http://maven.apache.org/>

APACHE SOFTWARE FOUNDATION. Apache Shiro. Disponível em: <http:// http://shiro.apache.org/>

BOOCH, G.; RUMBAUGH, J. ; JACOBSON, I. **The Unified Modeling Language User Guide**. 2ª Edição. Addison Wesley, 2005. 496p.

CARMICHAEL, A.; HAYWOOD, D. **Better Software Faster.** Prentice Hall, 2002.

COCKBURN, A. **Hexagonal Achitecture.** 2005. Disponível em: <http://alistair.cockburn.us/Hexagonal+architecture>

COLLINS, D. **Designing Object-oriented User Interfaces**. Redwood City: Benjamin/Cummings, 1995.

DATANUCLEUS. Data Nucleus. Disponível em: <http:// http://www.datanucleus.org//>

DIVISÃO DE BIBLIOTECA. **Diretrizes para Apresentação de Dissertações e Teses. Escola Politécnica da USP**. junho. 2006. 105p.

EVANS, E. **Domain Driven Design:** Tackling Complexity in the Heart of Software. 1ª Edição. Westford, EUA: Addison Wesley – Pearson Education, 2004. 529p.

FIRESMITH, D., **Use Cases: The Pros and Cons.** New York, EUA: R. Wiener, 1996.

FOWLER, M. **Analysis Patterns: Reusable Object Models**. 1ª Edição. Addison Wesley, 1996. 384p.

FOWLER, M. **Refactoring: Improving the Design of Existing Code**. 1ª Edição. Addison Wesley, 1999. 464p.

FOWLER, M. **Patterns of Enterprise Applications Architecture**. 1ª Edição. Addison Wesley, 2002. 560p.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. 1ª Edição. Addison Wesley, 1994. 416p.

HAYWOOD, D. **Domain Driven Design Using Naked Objects.** 1ª Edição. EUA: Pragmatic Bookshelf, 2009. 375p.

HAYWOOD, D. Introducing Apache Isis**. Software's Developer Journal**, v. 1, n. 1, p. 6-17, 2013.

ORACLE CORPORATION. MySQL Database. Disponível em: <http://www.mysql.com/:>

PAWSON, R.; MATTHEWS, R. **Naked Objects**. 1ª Edição. West Sussex, Inglaterra: J Wiley, 2002.

PAWSON, R. **Naked Objects.** 2004. 223 p. Tese (Doutorado) – Trinity College, University of Dublin, Dublin, 2004.

REENSKAUG, T. **Thing-Model-View-Editor**. Xerox Parc, 1979. Disponível em: http://heim.ifi.uio.no/~trygver/1979/mvc-1/1979-05-MVC.pdf13.

REENSKAUG, T. **Model View Controller**. Portland Pattern Repository. Disponível em: http://c2.com/cgi/wiki?ModelViewController

THIRUVATHUKAL, GEORGE K.; KONSTANTIN, LAÜFER. A Stroll Through Domain-Driven Development with Naked Objects. **Computing in Science Engineering**: **Scientific Programming.** P. 76 – 83. Maio/Junho 2008.